An algorithm to detect and communicate the differences in computational models describing biological systems

Martin Scharm[1,*], Olaf Wolkenhauer[1,2], Dagmar Waltemath[1]
1 Department of Systems Biology and Bioinformatics, University of Rostock, Rostock, Germany
2 Stellenbosch Institute for Advanced Study, Wallenberg Research Centre at Stellenbosch University, Stellenbosch, South Africa
∗ **E-mail: martin.scharm@uni-rostock.de**

Abstract

Repositories, such as the BioModels Database and the Physiome Model Repository support the reuse of models and ensure transparency about results in publications linked to those models. With thousands of models available, a framework to track the differences between models and their versions is essential to compare and combine models. Difference detection allows users to study the history of models but also helps in the detection of errors and inconsistencies. However, current repositories lack suitable methods to track a model's development over time. Consequently, researchers have problems to grasp the differences between models and their versions.

Focusing on SBML and CellML, we developed an algorithm to accurately detect and describe differences between versions of a model with respect to (i) the models' encoding, (ii) the structure of biological networks, and (iii) mathematical expressions. Our method is implemented in a comprehensive and open library called BiVeS. Our work facilitates the reuse and extension of existing models. It also supports collaborative modelling. Finally, it contributes to better reproducibility of modelling results and to the challenge of model provenance.

Our algorithm is the first tailor-made detector of differences between versions of computational models in standard formats.

1 Introduction

Modelling and simulation is a standard approach to investigate complex biological processes. A steadily increasing number of computational models are available from open repositories such as the BioModels Database [1] or the Physiome Model Repository (PMR2, [2]). These repositories provide the infrastructures necessary to collect and maintain model code and associated meta data. The distribution of models through these repositories accelerates collaborative research and encourages model reuse. The reuse of models improves the modelling workflow by reducing errors and saving time. Tracking the evolution of a model, providing information about changes in the model and its encoding, plays an important role in supporting the user [3]. The need of model version control has been emphasised repeatedly on several occasions [1, 3–5].

An example that demonstrates the need for tracking the evolution of models is the cell cycle. In 1993, Novak and Tyson published the first cell cycle model describing the M-phase control in *Xenopus oocyte* extracts and intact embryos [6]. The model representing these findings was first published in the BioModels Database in 2007 (release number 8) with the identifier BIOMD0000000107[1]. Along with 20 official releases of the database, the model has undergone numerous changes. Figure 1 shows the differences linked to one reaction of the model: The formation of Cdc2-cyclin dimers from Cyclin subunits and free Cdc2 monomers (Step 3 [6, Figure 1]). Changes between 2007 and 2013 only reflect regular updates of the internal encoding of the model in the BioModels Database. These modifications do not

[1] http://www.ebi.ac.uk/biomodels-main/BIOMD0000000107

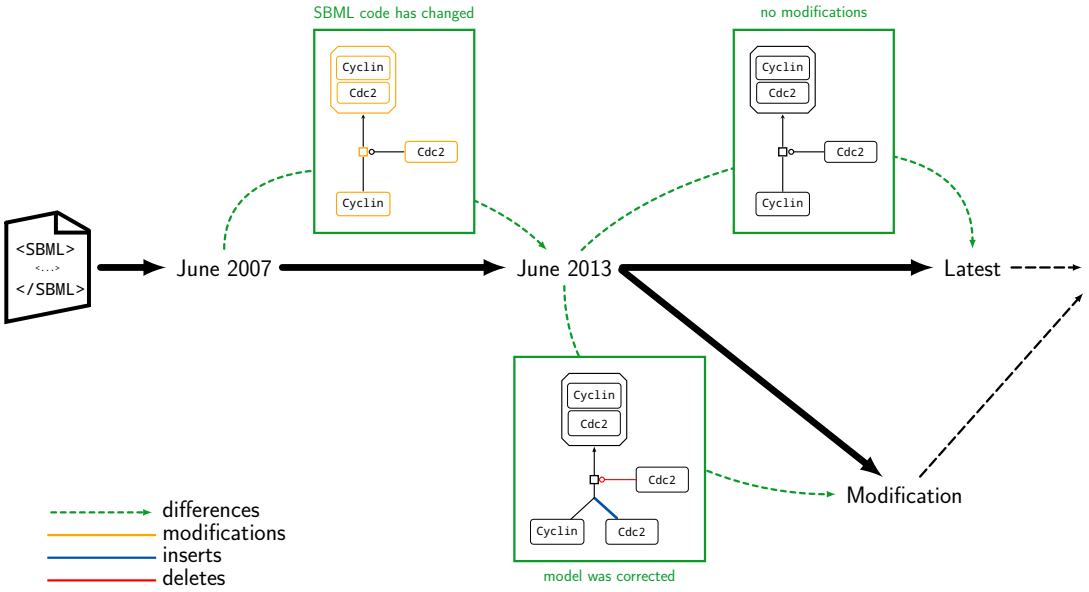


Figure 1. Sketch of a model's temporal evolution. Changes in a single reaction of Novak and Tyson's model with ID 107 in BioModels Database. The figure shows the differences between versions from June 2007 (release number 8), June 2013 (release number 25), and August 2014 (latest available version). The branch represents our own improvements from November 2013, which shall be merged with the BioModels Database version in the future. The green boxes visualise the differences between related versions.

relate to the biological meaning of the reaction. Along with these changes, numerous authors have build similar models or refined and extended the existing one. In this work, we focus on the detection and characterisation of such changes in models.

The relation between versions of a model from different releases can be considered the model's *history* (Figure 1). In the simplest case, the history follows a single line along the time axis. In practise, however, models are curated, refined and corrected. Open model repositories have their own well-defined pipelines for these kinds of modifications [7] which regularly lead to new versions of a model. After publication, models may be combined into larger networks, e. g. [8–10]. Also during model development, several alternatives are tested, leading to different paths in the history of a model. These so-called branches complexify the study of a model's history, especially when branches with different modifications shall be merged back into single model versions at a later time. For this reason, *difference detection* plays a key role in *model version control*. Under the term of model provenance, these issues are discussed in term of the seven W-questions: Who, What, Where, Why, When, Which, With (How)? [11–14].

In this paper we present a novel method for difference detection in models of biological systems. Our method for difference detection is implemented in a software library called BiVeS. It can immediately be used with existing model repositories and management systems. In addition, we showcase the capabilities of BiVeS in our web based tool for version control of SBML and CellML models, referred to as BudHat. BiVeS and BudHat demonstrate successful provenance of models for biological systems. Together, our work in model version control contributes to successful provenance of model-driven research in computational biology.

2 Methods

Our algorithm for difference detection takes two versions of a model and calculates the difference between them. We restrict ourselves to models encoded in XML representation formats. Our algorithm grounds on the XyDiff algorithm which focuses on efficiency in terms of speed and memory [15]. It distinguishes three major steps: (2.1) Pre-processing the XML documents, (2.2) Mapping the hierarchical structures, and (2.3) Post-processing the resulting mapping. Based on the identified mapping, the algorithm computes a delta (2.4). This delta can be converted into both machine readable (2.5) and human readable (2.6) formats. Each of these steps is described in the following sections.

2.1 Pre-processing of models encoded in XML

First the XML documents, which encode two versions of a model, are translated into an internal tree structure. For every node n in the tree a hash sum n_σ and a weight n_ω are calculated. With length(n) denoting the length of the text stored in n, the weight n_ω is determined by:

$$n_\omega = \begin{cases} 1 + \log(\text{length}(n)) & \text{if } n \text{ is text node,} \\ 1 & \text{if } n \text{ is leave,} \\ 1 + \sum\limits_{c \,\in\, \text{children}(n)} c_\omega & \text{otherwise.} \end{cases}$$

The weight of a node is thus always greater than the weight of its children. As such the weight represents the size of the corresponding subtree. The hash sum of a node n represents the signature of the subtree rooted at n. In the current version of our implementation we determine the hash n_σ of a node n by the SHA-2 sum of the concatenation of the node's tag name, its attributes and the hash sum of all its children. While n_σ unambiguously defines the subtree rooted in n, n_σ does not need to be unique among all nodes in the tree. Thus, if $n_\sigma = m_\sigma$ then the subtrees in n and m are identically equal. We will use these signatures to speed up the mapping of two hierarchical structures, as will be described in the next section.

2.2 Mapping model entities

To compare two tree structures T_1 and T_2 we use XyDiff's BULD algorithm, in which matchings are propagated bottom-up and only lazily down. It finds matchings between common large subtrees of the two documents and propagates these matchings [15]. Following the BULD algorithm, we distinguish the following four phases during the mapping stage.

Mapping by ID: First, if available, id attributes are being mapped, i. e., nodes in both documents sharing the same value of an id attribute are mapped onto each other. If many nodes are labelled with an id attribute, then a large number of mappings are already computed at this stage, and the following mapping procedure, which is computational harder, simplifies dramatically.

Bottom-Up propagation: Second, the initial mapping is propagated upwards into the trees. Therefore, the connections of a node's children are evaluated in a depth-first traversal of T_2. If a node n in T_2 is connected to a node m in T_1 then a mapping of parent(n) to parent(m) is suggested. The confidence of this suggestion equals n_ω and is therefore proportional to the size of n's subtree. If, in contrast, n is not connected, we examine the candidates which were previously suggested by the connections of n's children. Candidates which have a different tag name than n and candidates which already have a connection are neglected. Among the remaining candidates the algorithm chooses the one that received the best suggestions and connects it to n.

Top-Down propagation: Third, the algorithm makes use of the initially computed signatures and maps nodes of T_2 on nodes of T_1 which share the same hash value. A priority queue Φ is maintained to sort the nodes of T_2 based on their weights. Initially, Φ only consists of the root node of T_2. Unless Φ is

empty, the algorithm repeatedly removes node $n \in \Phi \subset T_2$ with the biggest weight, which represents the biggest subtree in the queue, and collects a set of mapping candidates $M \subset T_1$ with $\forall m \in M : m_\sigma = n_\sigma$. If M is empty all children of n are added to Φ and the loop continues with the next biggest subtree. Otherwise, the algorithm tries to find a node $m \in M$ for which there already exists a mapping between ancestors(m) and ancestors(n). As proposed by [15] the number of levels to chase the ancestry of both nodes depends on the ratio of n_ω to root$(T_2)_\omega$. Thus, for large subtrees we are willing to climb many levels in the tree to find a mapping of ancestors, but we might just examine firsthand parents in order to map leave nodes. If we are able to find such an $m \in M$ all nodes of the subtrees in m and n are mapped onto each other, just as the ancestors up to the discovered mapping.

Optimisation: Fourth, the algorithm improves the quality of the mapping by examining the network structure of T_1 and T_2 in a top-down approach. For every mapping $n \in T_2$ on $m \in T_1$ it compares unmatched children of n and m in order to find missed mappings. A distance matrix $M^{i \times j}$ is created with $M_{i,j}$ being the ratio of the number of differing attributes to the total number of attributes between the i-th child of n and the j-th child of m, or 0 if both nodes do not have any attributes. We assign ∞ to elements $M_{i,j}$ if the corresponding nodes already have a mapping, or if they do not share the same tag name. The algorithm evaluates the matrix greedily and adds new mappings up to a maximum distance of 0.9. Thus, nodes which have nothing in common will not be connected.

2.3 Post-processing of mappings

Whenever we have further knowledge of the data that is encoded in both trees we use it to revise the mapping. We therefore implemented further rules to exploit additional biological knowledge in order to capture the domain characteristics of the data being processed. Evaluating additional knowledge is a major factor why our algorithm outperforms standard XML diff algorithms.

Roughly speaking, we prohibit some network operations in the hierarchical tree of document nodes. In SBML encoded models for example, `listOf`-nodes are not allowed to change their parents. That means, if a `listOfModifiers` of T_1 is mapped onto a `listOfModifiers` of T_2 but their parents are not linked previously during the mapping procedure we drop this mapping. It is the same for nodes with a tag name of `speciesReference`, `modifierSpeciesReference`, `trigger`, `eventAssignment`, `delay` and `priority`. If the parents in the corresponding tree are not connected, which means their network in the XML documents differs, we remove the mapping from the set of operations. Unfortunately, these rules of course expand the set of operations in the delta later on, but we are willing to trade some minimality to increase the significance of produced deltas.

2.4 Computing the delta

A delta is a set of operations on entities (nodes or attributes, respectively) necessary to transform one document into another. We distinguish the following four types of operations which apply on entities of the corresponding XML tree:

insert if an entity is present in T_2 but absent in T_1

delete if an entity is present in T_1 but absent in T_2

move if a node is present in both documents, but either (1) the parents in the corresponding trees are not connected or (2) the parents are connected, but the sequence of their siblings has changed

update if the value of an attributes, a text node's content, or the tag name of a node was modified

While the set of move operations may only contain document nodes, the set of updates in general only consists of operations on attribute values. There is a single exception: The root nodes of both documents

are always mapped onto each other. Therefore we must include an operation which updates the tag name of nodes. However, we only support this operation for root nodes, though. Thus, internal document nodes will never occur in the set of updates and we will neglect this special case in the following.

After the mapping we distinguish two types of nodes: Mapped nodes and unmapped nodes. Unmapped nodes $n \in T_1 \cup T_2$ are nodes for which the algorithm could not find a matching node in the opposite tree. These nodes and their attributes correspond to either inserts or deletes, depending on their origin (T_2 or T_1, respectively). In contrast, mapped nodes are nodes for which the algorithm did find a matching node in the opposite tree. If the parents of such a mapping of $n \in T_2$ onto $m \in T_1$ are not connected, or if the sequence among their siblings has changed, then these nodes are included in the set of moves. Moreover, for each attribute $a \in \text{attributes}(n) \cup \text{attributes}(m)$:

- if $a \notin \text{attributes}(m)$ then a is included into the set of deletes
- if $a \notin \text{attributes}(n)$ then a is included into the set of inserts
- if $\text{value}(m, a) \neq \text{value}(n, a)$ the attribute is included into the set of updates

All other cases (nodes are mapped and occur at the same position in both trees; attribute values of mapped nodes are equal) do not call for an operation to transform T_1 into T_2 and are therefore not included in the delta.

2.5 Translating into machine readable XML

The resulting delta is then encoded in an XML document consisting of the four sections *deletes*, *inserts*, *moves* and *updates*. These sections contain three types of nodes:

- nodes with a tag name `node`, which describe operations on nodes
- nodes with a tag name `attribute`, which describe operations on attribute values
- nodes with a tag name `text`, which describe operations on text nodes

All these nodes have to carry an unique id attribute and, if available, must contain identifiers `oldPath` and `newPath` to unambiguously point to the corresponding nodes in T_1 and T_2, respectively. These identifiers are XPath[2] expressions, a language defined by the World Wide Web Consortium (W3C), to identify nodes in an XML document. Even if the delta is meant to be evaluated by machines, during the creation of those identifiers we make sure that they are as human readable as possible. In addition, `node` nodes may also contain the attributes:

- `oldParent` and `newParent` (XPath expressions) identifying the parents of the corresponding nodes
- `oldChildNo` and `newChildNo` (Integers) defining the position among their siblings, in order to encode *moves*
- `oldTag` and `newTag` (Strings) specifying the tag name of the corresponding nodes

Furthermore, `attribute` nodes may have three additional attributes:

- `name` defining the name of the corresponding attribute
- `oldValue` and `newValue` specifying the value of that attribute in T_1 and T_2, respectively

The generated delta is complete and, thus, it is invertible. That means, it contains all information necessary to transform T_1 into T_2, but it can also be used to obtain T_1 given T_2. An example is shown in Figure 3c.

[2] http://www.w3.org/TR/xpath/

2.6 Translating into human readable formats

To support the readability we currently export two different formats: (1) A text-based report; and (2) a graphical representation. The text-based report is a list of differences between the corresponding files. It contains all modified entities relevant for the biological model, such as parameters, species and reactions, and it contains the specific changes. The report can be generated in HTML, ReStructuredText, or Markdown. Markdown and ReStructuredText are easy-to-read plain-text markup languages, specifically designed to ensure a straightforward conversion to other markup languages, such as (X)HTML, doc(x), ODF, or LaTeX.

Moreover, standard formats usually allow for inclusion of information about the reaction network encoded in the computational model. If the models in question contain such information, we will first extract the reaction network of T_1 and translate it into an internal graph representation. Second, we obtain the reaction network of T_2 and put it as an overlay on top of the network of T_1, according to the previously computed mapping of the corresponding entities in T_1 and T_2. Subsequently, the graph is evaluated: We check whether nodes and edges originate from one or both documents and analyse what has changed in the corresponding tree nodes. To export this graph we developed translators that convert the internal graph representation to exchangeable graph formats, such as GraphML[3] or Dot[4]. These graphs can then be used in end-user applications, as described in the Results section.

3 Results

Model provenance and model version control are an important contribution to the reproducibility of model-driven results in biology. With a steadily increasing number of models generated in biology, there exists also an increasing number of models describing the same biological systems. Furthermore, existing models are continuously updated, generating new versions of a model. This generates a need for tools that can detect difference in models and support model version control.

We developed BiVeS, which incorporates a novel method that detects and highlights differences between two arbitrary versions of a model encoded either in the *Systems Biology Markup Language* (SBML, [16]) or in CellML [17]. Both formats are standard model representation formats used to exchange computational models describing biological systems. They are supported by a great number of software tools for modelling, analysing, visualising and simulating biological systems. There are in general three major processes which generate new versions: During the design phase of models, later on during curation in open model repositories, but also with updates of the SBML and CellML specification, or when errors are detected. Consequently, different versions of a model exist next to each other, as shown in Table 1.

	BioModels Database	CellML repository
#Models	489	2702
#Versions	4927	4481

Table 1. Number of models and versions in BioModels Database and CellML Model Repository.

BiVeS generates a mapping between model entities. Our approach first maps the entities in the entities in the two model files, and then identifies the changes that occurred. Figure 2 exemplifies the method, showing two versions of a toy model, whereby the reaction $C + D \rightleftarrows E$ (left) is updated to $D + H \rightleftarrows E$ (right). First, the model files are transformed into internal tree representations and prepared

[3] http://graphml.graphdrawing.org/
[4] http://www.graphviz.org/content/dot-language

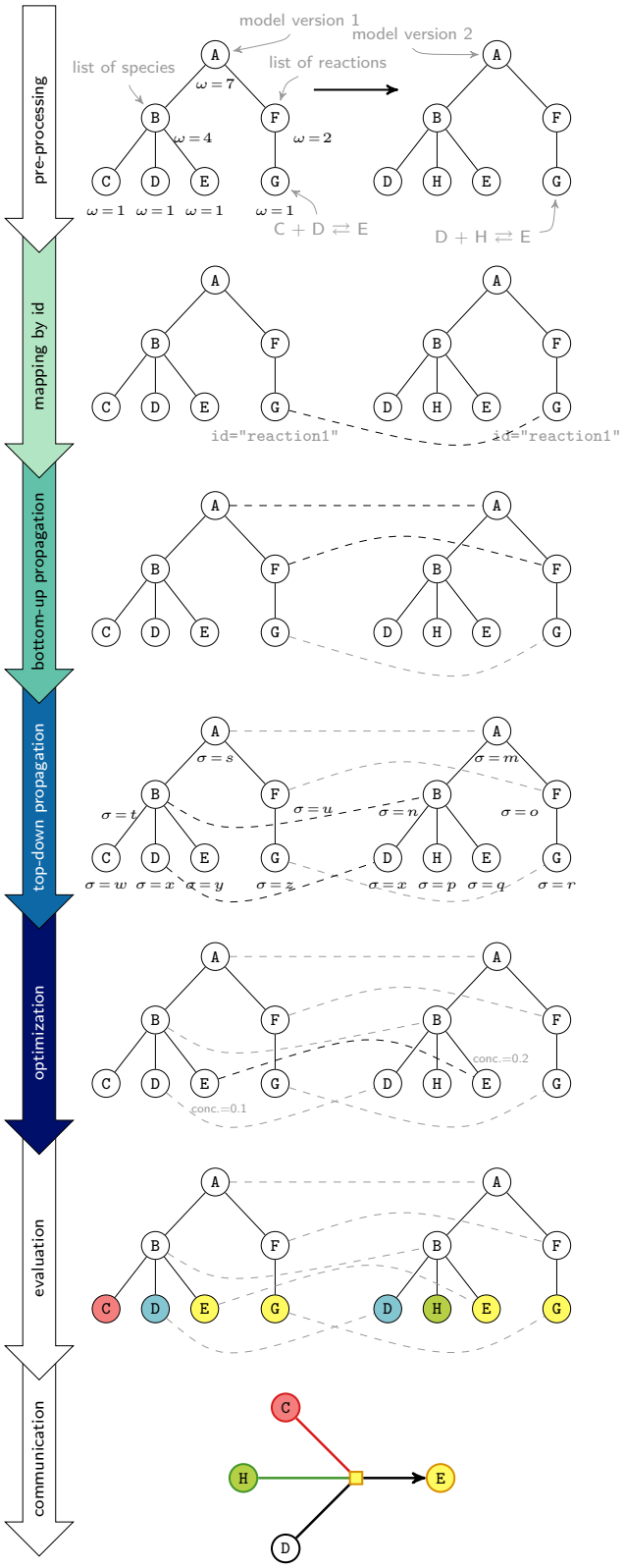


Figure 2. Schematic of the mapping procedure. The figure illustrates the procedure to communicate the differences between two versions of a model (row one) to the user (row seven). Nodes `A-H` represent single entities in the model documents. Dashed lines indicate mappings between the nodes. The values of σ and ω represent signatures and weights of nodes. They are calculated during pre-processing. The different colours of the nodes indicate modifications: Updates are yellow, inserts are green, deletes are red, and moves are blue.

for the subsequent mapping procedure (row one, *pre-processing*). The weights ω of nodes in the tree are computed according to the size of the corresponding subtrees. For example, the subtree rooted in `B` is larger than the subtree rooted in `F` and, thus, `B`'s weight is greater than `F`'s (namely $\omega = 4$ and $\omega = 2$, respectively). The mapping procedure starts in row two of Figure 2 with a *mapping by id*. Since the id attribute plays a key role and many elements do carry id attributes, the algorithm typically finds a large number of mappings at this early stage. In our example, only the identifiers of the `G`-nodes are identical (`id=''reaction1''`) and thus only a single connection is found. The *mapping by id* phase is followed by a *bottom-up propagation* (row three), which makes use of the parent-child relation of nodes in the trees: For nodes that are mapped already, there is a good chance that their parents also stem from each other. In the example, the mapping of the `G`-nodes is propagated towards the roots of the trees and the `A-F-G`-paths in both model versions are mapped. Afterwards, the algorithm tries to map subtrees with an equal signature (row four, *top-down propagation*). The signatures σ, which are computed in the *pre-processing* step, uniquely identify the subtrees. Here only the signatures of the `D`-nodes are equal ($\sigma = x$), which is why `D` is the only candidate for a mapping. Since the `D`-nodes originate from each other, as well as the `A`-nodes do, a mapping of the `B`-nodes is added. Following the propagation phases, the algorithm tries to connect unmapped children of mapped nodes (row five, *optimisation*). In our example, only the `B`-nodes have unmapped children: Nodes `C` and `E` in version 1 and nodes `E` and `H` in version 2 do not yet have partners. To find a mapping of these children, a 2×2 distance matrix is created. The elements in this matrix represent differences between the attributes of the corresponding nodes. The `E`-nodes only differ in the value of the concentration attribute. Changing the value of one single attribute in a species is a minor update and, thus, the nodes' distance is very small. In contrast, the nodes `C`, `E`, and `H` do not have anything in common. Consequently, the `E`-nodes will be mapped while `C` and `H` remain unmapped. Finally, the resulting mapping is analysed (row six, *evaluation*). For example, the algorithm detects that `C` was deleted, `D` was inserted and `E` was modified in version 2. The difference graph, as obtained when interpreting the results of the *evaluation* step, is shown on the bottom of Figure 2.

BiVeS brings difference detection to your favourite software tool. The above described algorithm for difference detection is implemented in a software library, BiVeS. It generates a list of differences between two model versions and exports it in several output formats, which include computer-digestible XML code or a graphical representation of differences. So far, BiVeS supports models encoded in CellML and SBML. The restriction on standard representation formats allows us to implement specific and powerful solutions.

One type of output are XML encoded, machine readable deltas, which describe the difference between two versions of a model (see Section 2.4). A remarkable feature of these deltas is their completeness. They can be inverted and composed [18]. That means, given one model version and the delta, the opposite version can be retrieved [4]. Another major feature of BiVeS is its capability of translating the delta into human readable formats. BiVeS summarises the model-related changes in a text-based report. This type of output is ideally suited to be integrated in other tools. Specifically, the report is either encoded in MarkDown, ReStructuredText or HTML. MarkDown and ReStructuredText are themselves already easy to read and can be converted to common markup languages. The report in HTML format is generated for convenience, e. g., to instantly display the changes on a web page. Figure 3b shows a sample report. Another notable feature of BiVeS is the encoding of differences between two versions of a model in standard graph representations enabling a subsequent visualisation. While BiVeS is itself not able to produce rendered graphical output, it exports different graphical notations, including GraphML[5], Dot[6], or JSON[7]. Armed with this, it is effortlessly possible to produce visualisations, as implemented in our demonstrator BudHat.

[5] graphml.graphdrawing.org
[6] graphviz.org
[7] json.org

BudHat showcases how BiVeS improves the understanding of a model's changes. Our web-based interface BudHat[8] demonstrates the described capabilities of BiVeS. BudHat contains a rudimentary user management and stores models in a database back-end. It calls BiVeS for the comparison two versions of a computational model and displays the obtained results in the web browser. The different visualisations that are possible in BudHat are shown in Figure 3. All figures show the difference between versions 2007-06-05 and 2013-11-03 of model BIOMD0000000107 in BioModels Database.

More specifically, BudHat provides access to:

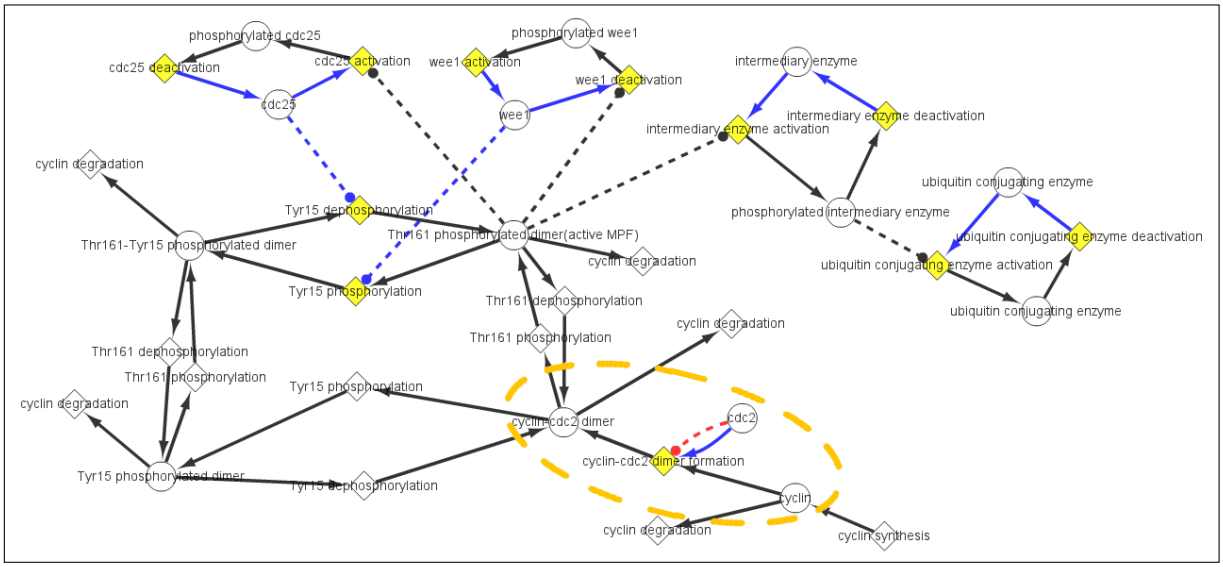
- the reaction network highlighting the changes, as shown in the upper part of Figure 3
- the HTML report of the changes, as shown on the left-hand side of Figure 3
- the delta encoded in XML, as shown on the right-hand side of Figure 3

Delta and report are directly passed to the web interface. But as stated above, BiVeS exports the graph representing the reaction network in an exchangeable format (in this case it is GraphML or JSON). Therefore, BudHat uses either CytoscapeWeb [19] or CytoscapeJS[9] to display the highlighted reaction network. From Figure 3a it is easy to see that the role of `cdc2` in reaction `cyclin-cdc2 dimer formation` has changed. In the former version `cdc2` modified this reaction, but this modification was deleted (deletion is highlighted by the red edge). Instead, in the updated version `cdc2` is one of the reactants for this reaction (insertion is indicated by the blue edge). Since this modification changed the reaction, the node representing the reaction is coloured in yellow. This approach makes it much easier to understand the differences, compared to a pure textual diff. Already for this small example, it would be much more effort to see and understand what happened to a model from the sources, or from the 1559 lines of output reported by Unix' diff.

Exemplifying difference detection for SBML models. We exemplify the advantages of our method using a reaction extracted from two different releases of model BIOMD0000000107 in BioModels Database. Figure 4 shows the SBML code of versions V_1 (June 2007, release 8) and V_2 (November 2013, release 26). BiVeS can now be used to compare these versions. Before starting the mapping procedure, it takes both model files and pre-processes them (see Section 2.1). For every node a signature and a weight are calculated. The signature uniquely identifies the subtree rooted in this node. A node's weight corresponds to the size of its subtree. The internal representations of the models can then be mapped (see Section 2.2). The algorithm first connects all nodes having an equal value of the id attribute (*mapping by id*). In this example, only the reaction nodes in both model versions carry an identical id attribute. Consequently, this step adds one mapping. Mapped nodes then trigger a mapping of the parent nodes (*bottom-up propagation*). In this case, the mapping of the `reaction` nodes entails a mapping of the `listOfReactions` nodes in both documents. Subsequently, BiVeS evaluates the previously calculated signatures and maps equal subtrees. Since nothing has changed in the subtrees rooted in the `kineticLaw` nodes, they are mapped onto each other. BiVeS then searches for mapped nodes which have children without a mapping (*optimisation*). Here, the node `reaction` is mapped, but its children are not, which triggers the creation of a distance matrix. Since the `annotation` nodes in both trees are equal the corresponding matrix element is 0. Same applies to the `listOfReactants` and `listOfProducts` nodes. All other elements in this matrix are ∞. Therefore, BiVeS adds three additional mappings. The optimisation phase recursively applies this procedure to all the children. First, it continues with a recursive mapping of the nodes in the `annotation` subtree. It stops at the `rdf:li` nodes, because these nodes have nothing in common but their tag names. The algorithm will not connect them. Second, BiVeS finds three unmapped children below the `listOfReactants` nodes (the `speciesReference` nodes representing Cyclin in V_1 and Cyclin/Cdc2 in V_2). While the computed distance between Cyclin nodes is $\frac{1}{3}$ (the value of one attribute differs), the distance between the Cyclin and

[8] budhat.sems.uni-rostock.de
[9] cytoscape.github.io/cytoscape.js

(a) Highlighted Reaction Network



(b) Report

Reactions

R19 (wee1 deactivation)	• wee1 → wee1_p • Modifiers: dimer_p(unknown);
R17 (cdc25 activation)	• cdc25 → cdc25_p • Modifiers: dimer_p(unknown);
R18 (cdc25 deactivation)	• cdc25_p → cdc25
R13 (Tyr15 dephosphorylation)	• p_dimer_p → dimer_p • Modifiers: cdc25(unknown)
R12 (Tyr15 phosphorylation)	• dimer_p → p_dimer_p • Modifiers: wee1(unknown)
R24 (ubiquitin conjugating enzyme deactivation)	• UbE_star → UbE
R23 (ubiquitin conjugating enzyme activation)	• UbE → UbE_star • Modifiers: IE_p(unknown);
R22 (intermediary enzyme deactivation)	• IE_p → IE
R21 (intermediary enzyme activation)	• IE → IE_p • Modifiers: dimer_p(unknown);
R20 (wee1 activation)	• wee1_p → wee1
R3 (cyclin-cdc2 dimer formation)	• cyclin + cdc2 → dimer • Modifiers: cdc2(unknown)

(c) XML encoded delta

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<bives type="fullDiff">
  [..]
  <move>
    <node id="8" newChildNo="4" oldChildNo="5"
      newParent="../listOfReactions[1]/reaction[1]"
      newPath="../kineticLaw[1]"
      oldParent="../listOfReactions[1]/reaction[1]"
      oldPath="../kineticLaw[1]"/>
  </move>
</bives>
```

Figure 3. Outputs as generated by BiVeS and available from BudHat. All three figures show the differences between versions June 2007 and November 2013 (cf. Figure 1). The reaction network (a) and the report (b) present the differences in a human readable format. The XML encoded delta (c) allows for further processing by computers. The modifications described in Figure 4 are highlighted in orange.

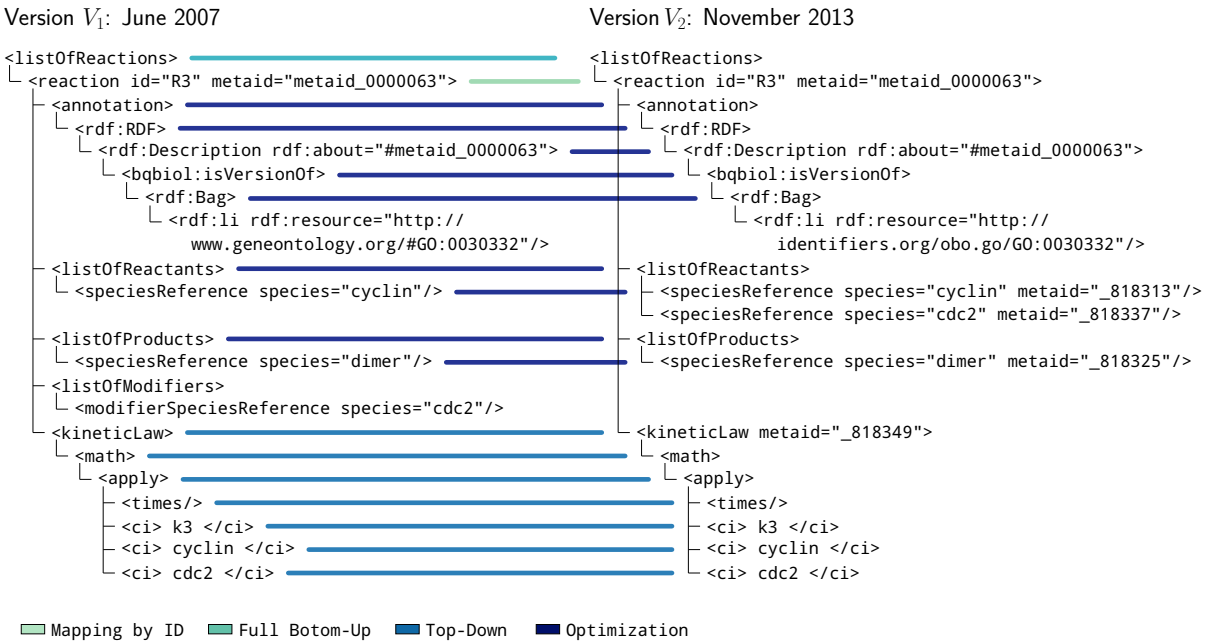


Figure 4. Example of a computed mapping. The figure shows the two versions V_1 and V_2 of the highlighted reaction from Figure 3. The differently coloured lines visualise the mapping of entities obtained when applying the four steps as described in Figure 2. Gaps between at rows on the left indicate inserted items, gaps on the right denote deleted items.

Cdc2 nodes is 1 (no matching attributes). Therefore, a mapping between the `speciesReference` nodes of the Cyclins is introduced. Third, a mapping between the `speciesReference` nodes (distance: $\frac{1}{3}$) below the `listOfProducts` nodes is detected. Eventually, the algorithm was able to find a mapping for most of the nodes. The unmapped nodes `rdf:li` and `speciesReference` (for species Cdc2) in V_2 correspond to inserts, and the unmapped node `rdf:li` as well as the subtree `listOfModifiers` correspond to deletes. An extract of the delta produced from the sample versions in Figure 4 is given in Figure 3c (XML is attached as S1 and S2). The shown delta describes the move of the subtree rooted at the fifth node below the first `reaction` (first node with a tag name of `kineticLaw`). This subtree was moved and became the fourth child of the same `reaction` node in version two (ll. 5).

The presented example is fairly small and straightforward. Yet it demonstrates how BiVeS easily outperforms Unix' diff. While BiVeS detects five deletes, eight inserts and one move, Unix' diff needs 19 deletes and 16 inserts. Apparently, Unix' diff is unable to meaningfully explain the modifications. The full example and the output of both tools can be found in the supplementary material.

4 Discussion

Reproducibility of model-based scientific results has gained increasing attention [20–27]. Indeed, the ability to reproduce results is a basic requirement for the advance of science [28]. However, the reuse of models requires the accessibility and comparability of models and their versions. Model provenance and version control enable the widespread use and application of models, saving time and costs during their development [3]. Model repositories have been working towards this goal for the past decade and provide access to computational models described in scientific publications. Support for version control, however,

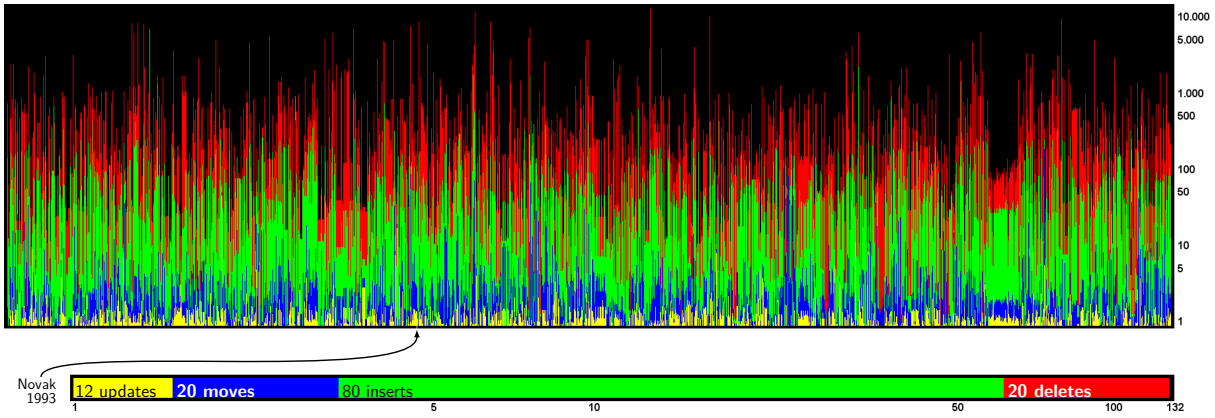


Figure 5. Nature of changes in the CellML Model Repository. Each bar corresponds to a version update. The y-axis shows the size of the delta on a logarithmic scale. The colours of each bar show the composition of the deltas: Deletes in red; inserts in green; moves in blue; and updates in yellow. The number of operations of each bar is plotted on a linear scale (as demonstrated for one update of the Novak 1993 model, from 5. July 2010). The prominent blue band represents the large number of move operations identified by BiVeS. All data is taken from CellML Model Repository, all releases till April 2014. The plot includes all transitions between versions with at least 10 operations.

is still limited. Existing implementations rely on standard version control systems and do not consider the specific requirements of modelling in the domain of computational biology.

BiVeS improves difference detection for your model versions. Standard formats describing computational models in biology are based on XML. Changes in versions of these models are typically computed with Unix' diff, which performs badly on XML documents because it uses a line based algorithm [29]. However, Unix' diff uses a line based algorithm [29] and therefore performs badly on XML documents. BiVeS, on the other hand, is designed to respect the characteristics of XML documents and to produce meaningful deltas. For example, it recognises the models' hierarchical structures (Figure 4). Furthermore, it ignores white spaces, such as indentation, which generally do not affect the model's behaviour. Finally, it ignores the specific order of attributes in an entity. Figure 6 compares the deltas as obtained by Unix' diff and BiVeS. To generate the data, we compared all versions of publicly available models from BioModels Database (top row) and the CellML Model Repository (bottom row), cf. Table 1. The boxplots on the left hand side of Figure 6 show that BiVeS needs fewer operations than Unix' diff to transfer one version of a model into another (avg. number of operations needed by BiVeS —Unix' diff: CellML Model Repository 349.5282—480.249; BioModels Database 434.825—1977.210). The scatter plots on the right hand side of Figure 6 confirm this trend.

One explanation, why BiVeS needs less operations than Unix' diff, is that BiVeS recognises moving entities. For example, if the sequence of reactions changes, Unix' diff merely detects insert and delete operations. Lets assume the reaction from Figure 4 gets moved to a different location in the document. Unix' diff would report up to 21 inserts and 21 deletes. In contrast, BiVeS recognises this modification as a single move operation. Indeed, move operations occur often, as shown in Figure 5. As the prominent blue band indicates, most version updates (72%) in the CellML Model Repository include at least one move operation. Moreover, 64% of these updates contain more than 5 moves.

In addition, we studied the ratio of direct and implicated operations. Deleting an entity, for instance, is a direct operation. The subsequent deletion of its attributes is triggered and, thus, an implicated

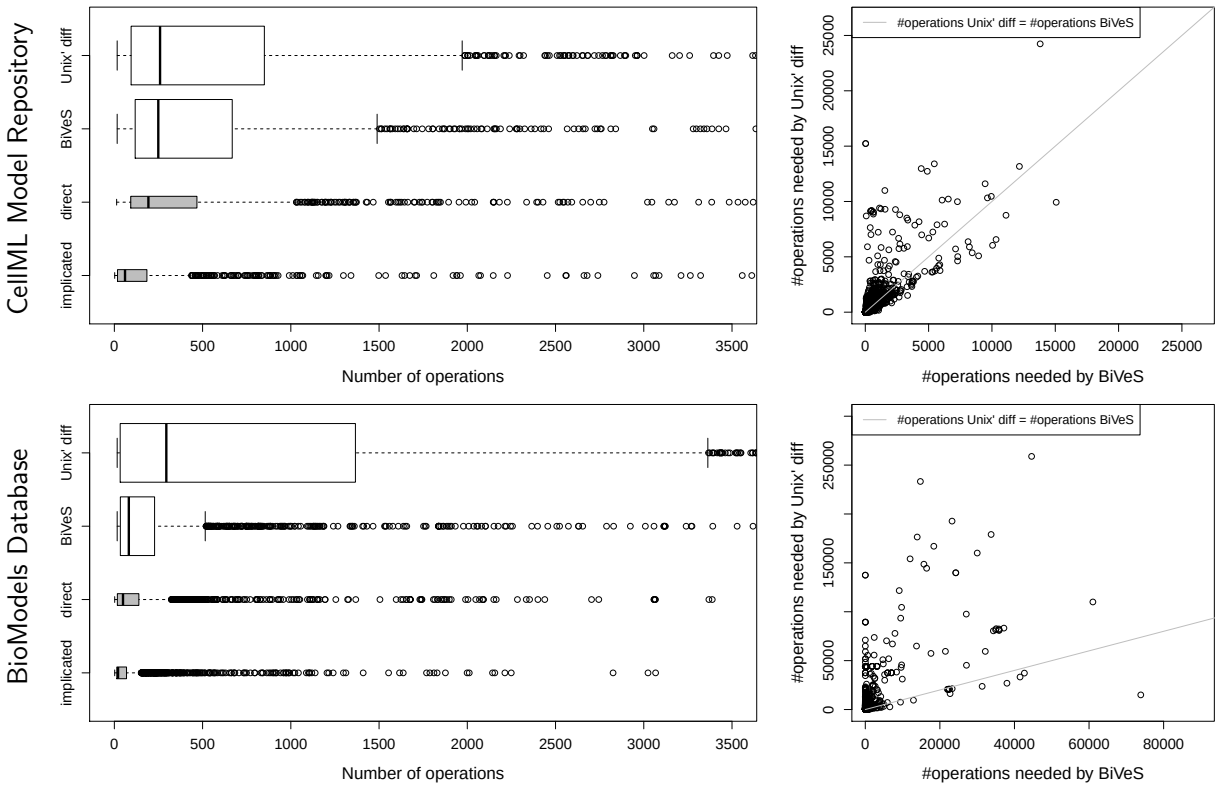


Figure 6. Comparison of operations needed by Unix' diff and BiVeS. The top row displays results obtained from models of the CellML Model Repository (12936 publicly available model transitions) and the bottom row displays results obtained from models of BioModels Database (9148 publicly available model transitions). On the left hand side bloxplots show the frequency of number of operations needed by Unix' diff and BiVeS. In addition, the operations as reported by BiVeS are divided into direct and implicated operations. On the right hand side, a scatterplot visualises the number of operations needed by Unix' diff and BiVeS for particular files.

operation. In both datasets, the number of direct operations is higher than the number of operations they implicate (Figure 6, bottom boxplots). This suggests that only a few modifications entail implicated operations.

BiVeS helps you grasping the changes. BiVeS filters identified differences and drops all but biologically and mathematically relevant changes. It thereby reduces the number of displayed changes to the ones that are helpful in deciding between two versions of a model. Furthermore, BiVeS produces reports and graphical representations of changes using open formats such as GraphML, HTML or Markdown, and thereby helps to communicate changes. BudHat was developed to demonstrate this. Figure 3a shows that the graphical representation supports the users in exploring the changes affecting the biological network. Additionally, a comprehensive list of changes is compiled into a human readable report, as shown in Figure 3b. Reports are particularly suitable for people interested in the details of mathematical changes. BiVeS' outputs can of course be used by other tools for further processing of results.

You can easily integrate BiVeS with your tools. BiVeS can be used in three different ways: First, the BiVeS Java library provides a smart API for comparison of model versions. The differences can then be obtained in various formats, as described earlier. This API is, for instance, used by BudHat, providing plenty of example code. Second, BiVeS is available as a web service to facilitate the integration with non-Java applications. The corresponding package can be installed on Java based web servers, such as Apache Tomcat[10]. The Functional Curation[11] project of Chaste [30], for example, uses the BiVeS web service to track the evolution of models uploaded to their system. Third, the library is shipped with a main class and, therefore, it can be executed on a command line. The data management platform SEEK [31], for example, implemented support for model version control calling BiVeS on a separate command line. The web site at sems.uni-rostock.de/bives offers further information about the three implementations, including examples, how-tos, the source code, and binaries of our tools. Researchers who do not work with one of the above data management tools may explore BiVeS' capabilities in our demonstrator BudHat. BiVeS currently supports SBML and CellML, but it could also be extended towards other XML-based exchange formats such as NeuroML [32] or PharmML [33]. Moreover, BiVeS could improve version control of simulation descriptions (e. g., differences between two simulation setups encoded in the Simulation Experiment Description Markup Language [25]) and associated results (e. g., encoded in the Numerical Markup Language[12]).

BiVeS improves the detection of differences between versions of models in SBML or CellML format. Returning to the seven W-questions from the introduction, BiVeS contributes to the *What* and *How* as defined in [12]. The *What* refers to content related events, such as modifications of parameter values in the model, and non-content related events, such as the upgrade to a new SBML version. In addition, BiVeS tells you *How* the *What* has changed. There is scope for further extensions to provide hypotheses for the *Why*.

5 Supporting Information

- **File S1. Model Version V_1 from Figure 4.** Extracted reaction `R3` from BIOMD0000000107, release number 8 (June 2007) of BioModels Database.
- **File S2. Model Version V_2 from Figure 4.** Extracted reaction `R3` from BIOMD0000000107, release number 25 (June 2013) of BioModels Database.
- **File S3. Unix' diff on S1 and S2.** Differences between S1 and S2 obtained by executing Unix' diff with `diff S1.xml S2.xml > S3.xml`.

[10] tomcat.apache.org
[11] chaste.cs.ox.ac.uk/FunctionalCuration
[12] http://code.google.com/p/numl/

- **File S4. BiVeS on S1 and S2.** Differences between S1 and S2 obtained by executing BiVeS with `java -jar BiVeS-fat.jar S1.xml S2.xml > S4.xml`.

Acknowledgments

Funding: This work has been funded by the German Federal Ministry of Education and Research (BMBF) as part of the e:Bio program SEMS, FKZ 031 6194. *Conflict of Interest:* None declared.

References

1. Li C, Donizelli M, Rodriguez N, Dharuri H, Endler L, et al. (2010) Biomodels database: An enhanced, curated and annotated resource for published quantitative kinetic models. BMC Systems Biology 4: 92.

2. Lloyd CM, Lawson JR, Hunter PJ, Nielsen PF (2008) The CellML Model Repository. Bioinformatics 24: 2122-2123.

3. Waltemath D, Henkel R, Hälke R, Scharm M, Wolkenhauer O (2013) Improving the reuse of computational models through version control. Bioinformatics 29: 742-748.

4. Saffrey P, Orton R (2009) Version control of pathway models using XML patches. BMC Syst Biol 3: 34.

5. Miller A, Yu T, Britten R, Cooling M, Lawson J, et al. (2011) Revision history aware repositories of computational models of biological systems. BMC Bioinformatics 12: 22.

6. Novak B, Tyson JJ (1993) Numerical analysis of a comprehensive model of M-phase control in Xenopus oocyte extracts and intact embryos. Journal of Cell Science 106: 1153-1168.

7. Chelliah V, Laibe C, Le Novere N (2013) BioModels Database: a repository of mathematical models of biological processes. Methods Mol Biol 1021: 189–199.

8. Gennari JH, Neal ML, Carlson BE, Cook DL (2008) Integration of multi-scale biosimulation models via light-weight semantics. Pac Symp Biocomput 13: 414–425.

9. Smallbone K, Simeonidis E, Swainston N, Mendes P (2010) Towards a genome-scale kinetic model of cellular metabolism. BMC Systems Biology 4: 6.

10. Krause F, Uhlendorf J, Lubitz T, Schulz M, Klipp E, et al. (2010) Annotation and merging of SBML models with semanticSBML. Bioinformatics 26: 421–422.

11. Goble C (2002) Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics. In: Workshop on Data Derivation and Provenance. Chicago, Illinois.

12. Ram S, Liu J (2010) Provenance management in biosciences. In: Trujillo J, Dobbie G, Kangassalo H, Hartmann S, Kirchberg M, et al., editors, Advances in Conceptual Modeling Applications and Challenges, Springer Berlin Heidelberg, volume 6413 of *Lecture Notes in Computer Science*. pp. 54-64. doi:10.1007/978-3-642-16385-2_8. URL http://dx.doi.org/10.1007/978-3-642-16385-2_8.

13. Moreau L, Groth P, Miles S, Vazquez-Salceda J, Ibbotson J, et al. (2008) The provenance of electronic data. Commun ACM 51: 52–58.

14. Davidson SB, Freire J (2008) Provenance and scientific workflows: challenges and opportunities. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. New York, NY, USA: ACM, SIGMOD '08, pp. 1345–1350. doi:10.1145/1376616.1376772. URL http://doi.acm.org/10.1145/1376616.1376772.

15. Cobena G, Abiteboul S, Marian A (2002) Detecting changes in xml documents. In: Proceedings 18th International Conference on Data Engineering. pp. 41 -52. doi:10.1109/ICDE.2002.994696.

16. Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, et al. (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19: 524-531.

17. Cuellar AA, Lloyd CM, Nielsen PF, Bullivant DP, Nickerson DP, et al. (2003) An overview of CellML 1.1, a biological model description language. SIMULATION 79: 740-747.

18. Marian A, Abiteboul S, Cobena G, Mignet L (2001) Change-centric management of versions in an xml warehouse. In: Proceedings of the 27th International Conference on Very Large Data Bases. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., VLDB '01, pp. 581–590. URL http://dl.acm.org/citation.cfm?id=645927.672205.

19. Lopes CT, Franz M, Kazi F, Donaldson SL, Morris Q, et al. (2010) Cytoscape web: an interactive web-based network browser. Bioinformatics 26: 2347–2348.

20. Gentleman R (2005) Reproducible research: a bioinformatics case study. Stat Appl Genet Mol Biol 4: Article2.

21. Laine C, Goodman SN, Griswold ME, Sox HC (2007) Reproducible research: moving toward research the public can really trust. Ann Intern Med 146: 450–453.

22. Mesirov JP (2010) Computer science. Accessible reproducible research. Science 327: 415–416.

23. Casadevall A, Fang FC (2010) Reproducible science. Infect Immun 78: 4972–4975.

24. Peng RD (2011) Reproducible research in computational science. Science 334: 1226–1227.

25. Waltemath D, Adams R, Bergmann F, Hucka M, Kolpakov F, et al. (2011) Reproducible computational biology experiments with SED-ML – the Simulation Experiment Description Markup Language. BMC Systems Biology 5: 198.

26. Waltemath D, Henkel R, Winter F, Wolkenhauer O (2013) Reproducibility of model-based results in systems biology. In: Prokop A, Csuks B, editors, Systems Biology, Springer Netherlands. pp. 301-320. doi:10.1007/978-94-007-6803-1_10. URL http://dx.doi.org/10.1007/978-94-007-6803-1_10.

27. Sandve GK, Nekrutenko A, Taylor J, Hovig E (2013) Ten simple rules for reproducible computational research. PLoS Comput Biol 9: e1003285.

28. King RD, Liakata M, Lu C, Oliver SG, Soldatova LN (2011) On the formalization and reuse of scientific research. J R Soc Interface 8: 1440–1448.

29. Myers EW (1986) An o(nd) difference algorithm and its variations. Algorithmica 1: 251-266.

30. Cooper J, Mirams GR, Niederer SA (2011) High-throughput functional curation of cellular electrophysiology models. Progress in Biophysics and Molecular Biology 107: 11 - 20.

31. Wolstencroft K, Owen S, du Preez F, Krebs O, Mueller W, et al. (2011) The SEEK: a platform for sharing data and models in systems biology. Meth Enzymol 500: 629–655.

32. Gleeson P, Crook S, Cannon RC, Hines ML, Billings GO, et al. (2010) Neuroml: A language for describing data driven models of neurons and networks with a high degree of biological detail. PLoS Comput Biol 6: e1000815.

33. URL http://pharmml.org.