BIOINFORMATICS ORIGINAL PAPER

Vol. 29 no. 6 2013, pages 742–748
doi:10.1093/bioinformatics/btt018

Systems biology

Advance Access publication January 18, 2013

Improving the reuse of computational models through version control

Dagmar Waltemath[1,*], Ron Henkel[1], Robert Hälke[2], Martin Scharm[1] and Olaf Wolkenhauer[1,3]

[1]Department of Systems Biology and Bioinformatics, University of Rostock, 18051 Rostock, Germany, [2]IT- and Media Center, University of Rostock, 18051 Rostock, Germany and [3]Stellenbosch Institute for Advanced Study (STIAS), Wallenberg Research Centre at Stellenbosch University, Stellenbosch 7600, South Africa

Associate Editor: Jonathan Wren

ABSTRACT

Motivation: Only models that are accessible to researchers can be reused. As computational models evolve over time, a number of different but related versions of a model exist. Consequently, tools are required to manage not only well-curated models but also their associated versions.

Results: In this work, we discuss conceptual requirements for model version control. Focusing on XML formats such as Systems Biology Markup Language and CellML, we present methods for the identification and explanation of differences and for the justification of changes between model versions. In consequence, researchers can reflect on these changes, which in turn have considerable value for the development of new models. The implementation of model version control will therefore foster the exploration of published models and increase their reusability.

Availability: We have implemented the proposed methods in a software library called *Biochemical Model Version Control System*. It is freely available at http://sems.uni-rostock.de/bives/. *Biochemical Model Version Control System* is also integrated in the online application *BudHat*, which is available for testing at http://sems.uni-rostock.de/budhat/ (The version described in this publication is available from http://budhat-demo.sems.uni-rostock.de/).

Contact: dagmar.waltemath@uni-rostock.de

Received on October 15, 2012; revised on December 14, 2012; accepted on January 6, 2013

1 INTRODUCTION

Modelling has become an integral tool for research in computational biology (Finkelstein *et al.*, 2004). The increasing impact of modelling for biology is reflected in the rapidly growing number and complexity of computational models of biological systems (in the following referred to 'models') (Henkel *et al.*, 2010; Li *et al.*, 2010). Current modelling projects such as the *Virtual Physiological Human* (http://www.vph-noe.eu/) require the usage of techniques for model coupling, merging and combination at different scales. Computational support is needed to manage models; to ensure model exchangeability, stability and result validity; and to foster communication between project partners.

*To whom correspondence should be addressed

Model representation formats standardize model encoding. Examples are the *Systems Biology Markup Language* (SBML) (Hucka *et al.*, 2003), *CellML* (Cuellar *et al.*, 2003) or *NeuroML* (Gleeson *et al.*, 2010). These formats represent a model's structure (e.g. the biochemical network) and allow basic annotation of the model to better convey a model's intention. For example, SBML developed an annotation scheme (Hucka *et al.*, 2010), which reuses the *Resource Description Format* (RDF) (Lassila *et al.*, 1998) and identifiers from the Minimal Information Required in the Annotation of Models (MIRIAM) (Le Novère *et al.*, 2005) Registry (Juty *et al.*, 2012). The valid description of a model is a requisite for its dissemination, but an additional descriptive layer is necessary to ensure direct result reproducibility. This layer is covered by the *Simulation Experiment Description Markup Language* (SED-ML) (Waltemath *et al.*, 2011), which is a format for the standardized encoding of simulation experiment setups. Further projects develop standard formats for result data [e.g. Numerical Markup Language (NuML) (http://code.google.com/p/numl/)] or graphical representations of models (SBGN) (Le Novère *et al.*, 2009). With an infrastructure at hand that provides modellers in computational biology with a rich set of model-related information, it is now time to think about integrated management solutions for models, their associated simulation experiments, result data, reference publications and so forth (Henkel *et al.*, 2012).

Curated model source code is published in model repositories, for example *BioModels Database* (Li *et al.*, 2010), the *Physiome Model Repository* (PMR2) (Yu *et al.*, 2011), *ModelDB* (Hines *et al.*, 2004) or the *Open Source Brain* (http://opensourcebrain.org). Open model repositories grant researchers access to published models. They provide a platform for model sharing and long-term storage, and they add support for model validation, curation and annotation of submitted models. Several publishers already ask for model source code to be made available along with the written paper (including Oxford journals, BMC journals, PLoS journals or the FEBS journal). They recommend upload of model code to open model repositories using standard formats and annotations. Consequently, open repositories are high-quality, reusable resources of models. However, one disadvantage of current systems is the unavailability of user-interpretable version information together with the subsequent lack of model histories.

Each model is subject to a characteristic set of changes that reflects the model's updates from its creation to curation, publication and later reuse in other contexts. One prerequisite for the study of a model's history is the availability of all significant model versions. Here, the model version control system (VCS) saves time in recapitulating the different modelling steps taken to build a model. This aspect also becomes relevant when publishing the work, teaching model design in courses or during model curation when curators need to discuss necessary model changes with authors before publication in a model repository. A VCS is capable of storing all existing versions of a model during its existence. Subsets of these versions can easily be filtered and displayed to the users. The level of detail depends on the specific application. For example, the developers of a model repository may choose to open all versions that reproduce the results in the reference publication. However, during the collaborative development of a model, all intermediate versions may be of interest for the project members.

The need for model version control has been previously discussed in research groups facing model evolution in computational biology (Beard *et al*., 2009; Cuellar *et al*., 2006; Hucka *et al*., 2010; Li *et al*., 2010; Miller *et al*., 2011). In general, VCSs such as Subversion (http://subversion.apache.org/) (SVN) or Mercurial (http://mercurial.selenic.com/) track every change made in a source document, along with information about who made the change and an optional log message containing information why a change has been made. PMR2 has recently demonstrated how Mercurial can be used to track and present model versions over the web (Miller *et al*., 2011). Once processed, the information that is recorded by a VCS enables users to study a model's past and to answer specific questions about the model (Fig. 1).

Modellers may investigate which parts of a model have been changed, and how these changes were justified. For example, changes in the model parametrization may be justified by a new publication. It is also interesting for a modeller to see how often a model has been changed and when it was last changed. This information indicates how recent a model is and whether it has been tested and used by the community. When reusing model code, which has been changed frequently in the past, the modeller might decide to confirm the model's validity on a regular basis. Simulation experimental setups are valid for a model at a certain time (or range of time). Changes in the model may directly affect the simulation results and therefore must be communicated and explained. In addition, the applicability of standard simulation experimental procedures, termed *functional curation* (Cooper *et al*., 2011), must be ensured for each single version of a model. Consequently, the availability of every model version used in a simulation experiment is a major requirement to ensure reproducibility of results.

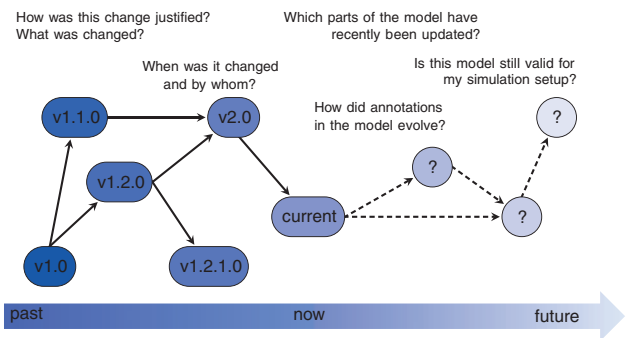

Fig. 1. Questions that a system for model version control can answer. Managing the versions of a model in a VCS is beneficial. Such a system tracks all relevant changes (What has changed? Why?). Questions about the model's history can be answered. In addition, researchers can study the likelihood of future changes and have a measure for a model's quality (Which parts of the model have recently been updated? By whom?). When models are linked to simulation experiments, consistency checks can be performed based on the detected model changes

In summary, the awareness of a need for version control led to the incorporation of technical solutions for model version control in model repositories. However, current approaches are appropriate for software code rather than model files. The standard algorithms that are to date implemented perform poorly on XML-encoded models. Existing solutions also miss support in aggregating version information about a particular model into a model history and interpreting changes. We argue that the implementation of tools that identify and highlight differences between models enables researchers to reflect on model changes, which in turn has considerable value for the development of new models. The present work discusses concepts for model version control and provides solutions to the previously mentioned problems. We concentrate here on models in XML-based standard formats. Our methods are document centric and therefore perform well on single-document models, e.g. SBML Level 2 models. However, in Section 4, we also show how multi-document models can be managed, e.g. CellML models with imported model components.

2 RESULTS

A model VCS should be tailored to existing model representation formats, which are typically XML and RDF based. It should furthermore reflect the temporal evolution of a model and present model changes to the users. In the following, current approaches to model version control are reviewed with respect to these requirements. Afterwards, a number of concepts for improved model version control are presented.

2.1 Current approaches to model version control

Two major approaches to model version control are currently taken: documenting changes directly in the model file *versus* addressing version control inside the respective model repositories. Both approaches are outlined in the following.

2.1.1 Version control inside model representation formats The different standardization communities offer means to store version information inside the respective model representation format. For example, the CellML metadata specification (Cuellar *et al*., 2006) distinctively stores 'trivial changes' (e.g. error corrections during the translation of a model to CellML) and 'substantial changes' (e.g. creating a new revision of a model) (Beard *et al*., 2009). SBML implements the model history as an additional element of the language itself, which can be attached to any SBML element. The main objectives for the SBML

history element are the provision of information on the creators of the encoding and the provision of modification dates (recording the creation of the model constituents and their subsequent changes).

In general, the storage of version information inside the model representation ensures that all changes applied to a model are shipped together with the model code. However, the disadvantage of this approach is that the description of changes inside the model file is in itself already a change of model code. Consequently, tools that are in principle capable of calculating differences between two files will incorrectly identify these updates as changes, even though the code updates do not reflect changes in the model. Furthermore, the history concept in representation formats is rather limited and not fully supported by software tools. The quality of the history is highly dependent on manual additions provided by the model authors and therefore more often than not incomplete.

2.1.2 Version control inside model repositories As an alternative, model repositories may implement support for version control. PMR2, for example, builds on a distributed VCS (Miller *et al*., 2011) based on Mercurial. Instead of hosting one central repository, each developer may keep a local instance. Changes can then be pulled from and pushed back to PMR2. Once submitted, models can subsequently be imported as a particular revision of a workspace. The workspace revisions also enable authors to review the change sets of models. Change sets are the collection of all differences between two versions of a model. CellML offers continuous version control at all times using exposures. An exposure (snapshot) is a publicly viewable presentation of a particular revision of a model. It may contain any type of generic text-based files (e.g. experimental setups, documents or simulation experiment descriptions). ModelDB also provides preliminary version support for a subset of the available models. Similarly to the CellML model repository, it uses an implementation of Mercurial. Here, a collection of models includes all revisions and associated versions of all models, and a log is available. This log contains the age of each model version, the submitter and a short description. It can be explored in multiple ways, including a tag-based view, a graph-based view or a branch-based view. BioModels Database uses the common non-distributed VCS SVN. The original file (i.e. the first uploaded version of the model in its original format) is provided on the model overview page, but also the SBML model version at each release point is available. BioModels Database distributes new versions of a model approximately four times a year with each release of the database. To allow users to retrieve and compare different versions of a model and its annotations in the future, BioModels Database developers plan to extend the functionality of their VCS (Li *et al*., 2010).

Recapitulating, an integrated history inside a model repository ensures the availability of all model versions at all times and the consistency of model files. However, only a complete checkout of a model, for example using PMR2, keeps the link to previous versions. Once a user downloads a model file from a repository, he loses this link. Furthermore, to date, no software tool allows displaying a model's history in a particularly useful format; neither can model histories be meaningfully interpreted.

2.2 Concepts for improved model version control

We present here solutions to selected problems identified in the analysis of current systems for model version control in computational biology. Our approach includes XML-aware difference detection, change transparency and justification of changes. These concepts will be explained in detail in the following paragraphs.

2.2.1 XML-aware difference detection VCSs that are used in software development are designed for programming code and text documents. Examples of such systems are SVN, Git or Mercurial. They implement a line-based algorithm that relies on the Longest Common Subsequence (LCS) (Hirschberg, 1977). LCS identifies changes between two files by matching the LCS of characters in them. A major concern with existing approaches to model version control is the choice of algorithm for difference detection. The set of operations that describes the transformation of a file version into its successor is denoted as diff. A diff between two versions of a model must contain all changes in terms of operations and yet be complete, minimal and interpretable. At a first glance, model code is in the same way semi-structured as is source code; both contain structured and free-text elements. In fact, BioModels Database stores the different releases of model files in an SVN (Li *et al*., 2010), and the CellML model repository implements a Mercurial system (Miller *et al*., 2011). Both solutions are LCS based. However, we tested the applicability of several methods implementing the LCS algorithm on difference detection in models. We find that line-based algorithms perform poorly in the difference detection of models. The main reason is that LCS does not respect the XML structure (Rönnau *et al*., 2005) that constitutes model files. The XML format may already change when a model is loaded and then exported by a simulation software, e.g. leading to changes in the indentation of XML code or to the reordering of XML elements. Model code is often automatically generated in software tools (e.g. the SBML representation of a model in the biochemical pathway simulator COPASI). Each software tool has its own preferred way of representing the model code, sorting the occurring XML elements or breaking lines in the XML code. Furthermore, people tend to open downloaded models in text editors, which reformat the XML code automatically. These common changes are detected by the LCS algorithm, but they are in fact irrelevant for the model's history and would be neglected by entity-based algorithms. In other words, although being successfully used for source code version control, LCS is not suitable for XML version control (Chawathe *et al*., 1996). Moreover, Krause *et al*. (2010) already mentioned that the LCS-based systems do not offer sufficient branching and merging options for XML-encoded data; these features have to be implemented on top of existing tools.

Owing to the format of existing model representations, we propose to reuse algorithms for difference detection in XML documents. Some systems are available that specifically work for XML documents, e.g. (Chawathe *et al*., 1996; Rönnau *et al*., 2005; Rosado *et al*., 2009). They build on standard XML diff algorithms such as *Diffxml* (http://diffxml.sourceforge.net/), *XyDiff* (http://leo.saclay.inria.fr/software/XyDiff/) or its Java-based variant *JXyDiff*. These algorithms can be used to detect differences in models. However, merging XML

file versions, in general, is an ongoing research topic in information systems (Rönnau *et al*., 2009). We demonstrated the use of the XyDiff algorithm in our library for difference detection, *Biochemical Model Version Control System* (*BiVeS*) (see Section 3).

2.2.2 Change transparency Entity-based and XML-aware algorithms reliably identify the differences between two versions of a model. This information must be stored in a way that can be interpreted and explored easily by researchers. For example, a model's parameters might have been updated owing to new scientific findings, and it is then important to know which parameters were changed and what the old and new values are. However, these updates in model repositories are to date not propagated to the user. We argue here that there should be a way for modellers and curators to identify the particular versions of a model that exist. Moreover, users should be able to easily identify who contributed to a particular model version. Multi-partner projects can result in models that originate from several different partners, e.g. the metabolic yeast model (Herrgård *et al*., 2008). In these cases, a mechanism for version control can identify and manage the single contributions of authors for each version of the model.

Another reason to propagate model changes is a model's use in simulation experiments. A SED-ML description does not necessarily contain the models, which were used in the experiment, but it often links to the models by an unambiguous identifier. For example, the reference to the Repressilator model in BioModels Database is identifiers.org/biomodels.db/BIOMD0000000012. An update of that model in BioModels Database may lead to invalid or unexplainable simulation results when executing the SED-ML file, e.g. if an observed species was deleted or renamed in the model code. When executing the SED-ML file, which re-uses that model, it is therefore crucial to alert users about the changes.

In summary, it is not only important to inform users on a model update *per se* but also to provide information about what has been updated. These requirements can be realized with the aforementioned diff files, which store all detected differences between two model files. Here, the XPath concept is used to unambiguously identify the location of each such difference. The diff file itself can again be encoded in XML. An alternative solution was suggested by Saffrey and Owen (2009) who proposed to store XML patches in the context of version control for SBML-encoded pathway models. A patch contains information about how to convert the original model into its successive version. This approach proposes a converter, whereas the aim of our work is an explicit listing of all changes in a model.

2.2.3 Justification of changes All relevant changes should be justified with references to a reliable source explaining reasons for that change. The understanding of changes leads to a better confidence in the model and thereby encourages researchers to reuse existing models. For example, a model curator should be able to annotate the update of model parameters with the publication providing ground for the new values. If a model in a repository is updated owing to new biological insights, an explanatory link to a public database can be provided. Also, an updated kinetic parameter could be justified with a link to the SABIO-RK database (Wittig *et al*., 2006), which contains information on curated reaction rates, parameters and their experimental conditions.

A technical mean to encode such justifications are references to external resources such as controlled vocabularies or ontologies. MIRIAM annotations, which are already used for model annotation, could also be used for difference annotation. Table 1 shows a possible coarse-grained classification of types of model changes to exemplify the use of annotations for the characterization of model changes. The draft version distinguishes the location of changes (XML *versus* annotation) from the type of change (typographic corrections *versus* changes in the underlying biology). All changes can be mapped on specific XML operations (update, insert, delete, move).

Each difference between two versions of a model can be annotated with terms from this classification, and several annotations might apply to one change. For example, each change occurs at a particular position in the XML file and therefore can be marked with a term from the Location branch, i.e. attribute, element or value change. In addition, a change that is annotated with the terms parameter (from the Type branch) and update (from the Operation branch) represents an update in a parameter value of a modelled entity.

When implementing a history-aware software tool in a model database, the aforementioned classification can be used to display only relevant changes (with respect to a particular application), e.g. during curation. The classification can, in addition, be used for automated reasoning. For example, a new version of a model that only contains annotation updates does not need to undergo a validity check on associated simulation experimental setups, as the model structure is not affected by the annotation updates. To implement reasoning on model changes, we plan to reuse existing methods for reasoning on SBML models, such as the ones developed by Hoehndorf *et al*. (2011).

3 IMPLEMENTATION

We implemented the aforementioned requirements in our software library *BiVeS*. The focus is on models encoded in SBML format, but the methods are similarly applicable to other XML-based model representation formats such as CellML, NeuroML and comparable. To use *BiVeS* with other model representation formats, the model parser needs to be extended, for example by integrating the CellML Application Programming Interface (API) (Miller *et al*., 2010). The restriction on XML-based formats allows us to reuse algorithms for XML version control. *BiVeS* is based on the existing XyDiff algorithm.

Table 1. Controlled vocabulary for change classification

Location		Type	Operation
XML	Annotation	Mathematics	Update
Element	Qualifier	Biology	Insert
Attribute	URI	Parameter	Delete
Value		Typo	Move
		...	

The focus on SBML is rooted in the high number of models available from BioModels Database for testing. We downloaded all existing versions of the curated models that were available from BioModels Database releases (ftp://ftp.ebi.ac.uk/pub/databases/biomodels/releases/). We subsequently ran *BiVeS* to find changes between the single SBML model files. Examples for models in different versions and their calculated and visualized differences are available on our testing platform called *BudHat* at http://sems.uni-rostock.de/budhat.

4 DISCUSSION

All relevant changes applied to a model should be tracked and they should be fully listed and documented. Users need to know what has been changed in a model, by whom, why and when (Beard *et al.*, 2009). Surprisingly, these requirements are not yet covered by existing systems. In this section,we discuss the improvements of our system over existing solutions and we point to open issues.

Improving model version control: To date, model files are rarely accessible by the users of model repositories in formats other than the original model file, or the processed, curated model file. For example, SBML files from BioModels Database are available for every release point. However, model code might change between two releases. These changes are not traceable by the community. Furthermore, the current BioModels Database releases are snapshots of the repository; they do not relate different versions of a model. Also, the concept of a model history inside SBML is not intuitive; different steps must be taken to reconstruct the history of a model in terms of curation and modification dates. The types of changes are not encoded in SBML models. The model database PMR2 already provides a system for model version control. The repository stores exposures and Mercurial logs, which represent different versions of a model. However, the interpretation of changes remains difficult, and the system lacks a visual representation of the model history.

A consolidated view of all these factors indicates that a sophisticated model VCS is missing in current software tools. Here, we propose a system that identifies and classifies the changes in a model. Instead of using built-in line-based difference detection (as used by SVN), we provide the XML aware difference detector *BiVeS*. Figure 2 shows how a model history can be maintained, and differences between two model versions can be detected and then visualized in *BudHat*. Such a graphical representation allows users to quickly grasp the difference between two model versions and to understand the evolution of a model over time.

However, a number of open questions remain for future investigations, despite the improvements introduced here.

Leaving the choice to the user: Providing a user with the differences between model versions enables him to actively decide whether to use an updated version of a model. For example, on discovering that a model has solely been updated to a new version of SBML, a user may decide to keep the older version of the model until his software is updated to support the new format specification. In this case, the model remains biologically valid. A decision for a model version requires a user to be aware of the model's change compared with its preceding version. This decision is supported by a visual representation of changes, as we have prototyped in BudHat (see again Fig. 2). Ongoing research

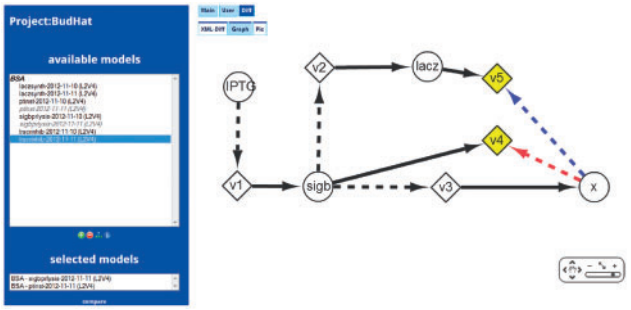

Fig. 2. The *BudHat* tool for model version control. The figure shows our prototype implementation of *BudHat*. Users can login to create their own history of model files (private or shared). The list of available models is shown on the left hand side of the figure. Two models from that list can be chosen and compared. The graphical representation shows the differences between these two model files. The differences are calculated by the *BiVeS* library. The result can be viewed either as an XML description of changes, or be displayed graphically, using an adaptation of the Cytoscape web tool. All changes in the network structure are colour coded as follows: new elements in blue, deleted elements in red and updated elements in yellow. A second feature is the graphical representation of a model's history (not shown in this figure). It provides a quick overview of existing versions of a model in *BudHat*

in the field of data and network visualization can contribute to this problem.

Models must be unambiguously identifiable: Reproducibility of results, based on a simulation setup, requires the simulated models to remain accessible. SED-ML files use the identification scheme provided by the MIRIAM registry (http://www.ebi.ac.uk/miriam/main/) to link to the SBML models involved in a simulation. However, it is currently impossible to address a particular version of a model. Only the latest version of a model is addressable. For example, the identifier for the Repressilator model in BioModels Database is identifiers.org/biomodels.db/BIOMD0000000012. The identifier is the only entry point to a set of versions of the SBML model, and no mechanism exists to reference a particular one of these. This scheme potentially leads to errors in the related simulation descriptions, as the person executing a SED-ML file cannot know whether the model has been updated since its last execution. Sometimes, software may not even be able to run an updated version of a model. Consequently, a scheme for the unambiguous identification of model versions is required. The developers of PMR2 already demonstrated how unambiguous links for each revision (exposure) of a model allow to reference single versions of a model (Miller *et al.*, 2011). This concept could provide a valid solution for other model repositories too.

Version control for multi-document models: Our approach to model version control is document centric, meaning it handles only single-document models. However, with the development packages in level three of SBML, different kinds of multi-document models can be generated. Also, in CellML, the concept of importing model components results in multi-document models. As BiVeS is document centric, it cannot *per se* handle multi-document models. We envision two different solutions here: BiVeS could be used as a stand-alone tool to detect the differences between model versions. In this case, the

multi-document models must be flattened (i.e. merged into one single file). A second solution is to combine the VCS with a database in the background handling multi-document models. Consequently, the problem is shifted to the storage layer of the model management system. We have already shown in our previous publication that SBML models can be transformed into a graph-based representation and then be managed by a graph database (Henkel *et al.*, 2012).

This approach can be extended to enable version control for multi-document models (Fig. 3): using a graph structure allows for relating models and model constituents. An import relation in the database links the model to constituents that are defined outside the model file, thereby representing multi-document models. Subsequent changes in one model can be propagated to all models reusing the updated components as a constituent. Figure 3 shows an example of a multi-document model and outlines how the relations between database nodes can be used for checks on updates in single models.

Version control for systems of models: All ideas in this article discussed the identification of changes between two versions of a single model, and the previous section outlined solutions to handle models that are composed of more than one document. However, even further information can be gained when looking at the evolution of models for a particular biological system. For example, a significant number of models that are based on the early encoding of Tyson's cell cycle model (Tyson, 1991) have been submitted to and published in model repositories. Looking at the different cell cycle models, how they relate to each other, how the single models evolved and how parts of models were merged into new models requires not only a version control of single XML files but also of a collection of models. One approach for addressing the problem of linking entities across model files is the use of annotations, e.g. using the Computational Neuroscience Ontology [CNO (http://purl.bioontology.org/ontology/CNO)], which contains a branch on model classification. However, the definition of similarity measures for changes across models is still an open research question.

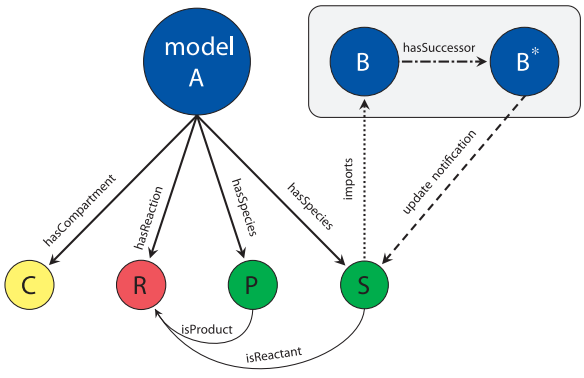

Fig. 3. Version control for multi-document models. The figure shows an example for multi-document models: Here, models A and B are stored, and model B has a successor B*. Furthermore, component S in model A imports model B. When managed in a repository, the information about occurring changes in model B can also be propagated to model A, which is using model B. More precisely, when version B* is created, a notification link to model A's component S can be created in the database. Thereby, the repository marks component S as changed, resulting in further updates if model A itself has been used as a component of another model. As it is not possible for now to rule whether changing model B to B* has an impact on model A, it is up to the user to resolve possible conflicts

What is a new model—and what is not? Models may exist in different versions over time, and they are generated by different research groups. Furthermore, researchers may produce a number of instances of a model with differing parameterizations (Finkelstein *et al.*, 2004). The fact is that models must be accessible for re-use and to foster result reproducibility. Both reduce costs and effort during model development. Consequently, questions such as how to distinguish model versions from each other and how to separate models from simulation setups to avoid the unnecessary redefinition of models as 'new models' must be discussed in the future.

5 CONCLUSION

Current technical solutions to the problem of model version control do not allow users to access, study, compare and visualize different versions of a model. Here, we have identified the following requirements for an improved system:

(1) An XML-aware algorithm for difference detection should be used: Moving from LCS to entity-based algorithms for XML version control is a first step towards a more efficient model VCS. Entity-based change detection is minimal, as it neglects XML formatting changes. We consequently propose the extension of XML-aware algorithms for difference detection in models.

(2) All changes should be transparent to the user: It is essential to know *which* parts of a model have been changed and *how*. This information can be encoded in a list of differences for pairs of two model versions. We consequently suggest to provide users with a visual representation of changes between model versions.

(3) Justification should be given for each change: It is important to provide justification for each relevant change in a model file, thereby encoding *why* something was changed and *by whom*. Entity-based difference detection enables partially automatized annotation of changes and their classification. We suggest here to establish tools for the semi-automatic annotation of model changes with terms from a controlled vocabulary.

These requirements should be respected in software tools that offer model management. We have implemented a library for model change detection, *BiVeS*, which can be integrated with existing databases or VCSs. Our online application, *BudHat*, furthermore exemplifies the envisioned functionality of a model VCS.

ACKNOWLEDGMENTS

The authors would like to thank the COMBINE community for valuable discussions on model version control during the COMBINE 2012 meeting. D.W., R.He. and O.W. identified the requirements for model version control. D.W. and R.He. developed the concepts. R.Hä. originally implemented the

BiVeS library. M.S. extended *BiVeS* and implemented the online application *BudHat*.

Funding: This work has been funded by the German Federal Ministry of Education and Research (e:Bio program).

Conflict of Interest: none declared.

REFERENCES

Beard,D.A. *et al.* (2009) CellML metadata standards, associated tools and repositories. *Philos. Transact. A Math. Phys. Eng. Sci.*, **367**, 1845–1867.

Chawathe,S. *et al.* (1996) Change detection in hierarchically structured information. *ACM SIGMOD Rec*, **25**, 493–504.

Cooper,J. *et al.* (2011) High-throughput functional curation of cellular electrophysiology models. *Prog. Biophys. Mol. Biol*, **107**, 11–20.

Cuellar,A. *et al.* (2006) The CellML metadata 1.0 specification. http://www.cellml.org/specifications/metadata/cellml_metadata_1.0 (30 January 2013, date last accessed).

Cuellar,A.A. *et al.* (2003) An overview of CellML 1.1, a biological model description language. *Simulation*, **79**, 740–747.

Finkelstein,A. *et al.* (2004) Computational challenges of systems biology. *IEEE Comp.*, **37**, 26–33.

Gleeson,P. *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+.

Henkel,R. *et al.* (2010) Ranked retrieval of computational biology models. *BMC Bioinformatics*, **11**. 423+.

Henkel,R. *et al.* (2012) Considerations of graph-based concepts to manage computational biology models and associated simulations. In: Goltz,U. *et al.* (ed.) *Workshop on data in the life sciences*. "INFORMATIK 2012", Lecture Notes in Informatics **208**, 1545–1551.

Herrgård,M. *et al.* (2008) A consensus yeast metabolic network reconstruction obtained from a community approach to systems biology. *Nat. Biotechnol.*, **26**, 1155–1160.

Hines,M.L. *et al.* (2004) ModelDB: a database to support computational neuroscience. *J. Comput. Neurosci.*, **17**, 7–11–11.

Hirschberg,D. (1977) Algorithms for the longest common subsequence problem. *J. ACM*, **24**, 664–675.

Hoehndorf,R. *et al.* (2011) Integrating systems biology models and biomedical ontologies. *BMC Syst. Biol.*, **5**, 124.

Hucka,M. *et al.* (2003) The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, **19**, 524–531.

Hucka,M. *et al.* (2010) The Systems Biology Markup Language (SBML): Language specification for level 3 version 1 core. http://iweb.dl.sourceforge.net/project/sbml/specifications/Level%203%20Ver.%201/sbml-level-3-version-1-core-rel-1.pdf (30 January 2013, date last accessed).

Juty,N. *et al.* (2012) Identifiers.org and MIRIAM Registry: community resources to provide persistent identification. *Nucleic Acids Res.*, **40** (D1), D580–D586.

Krause,F. *et al.* (2010) Annotation and merging of SBML models with semanticSBML. *Bioinformatics*, **2**, 421.

Lassila,O. *et al.* (1998) Resource description framework (RDF) model and syntax specification. W3C Recommendation. http://www.w3.org/TR/REC-rdf-syntax/ (30 January 2013, date last accessed).

Le Novère *et al.* (2005) Minimum Information Requested In the Annotation of biochemical Models (MIRIAM). *Nature Biotechnol.*, **23**, 1509–1515.

Le Novère,N. *et al.* (2009) The systems biology graphical notation. *Nature Biotechnol.*, **27**, 735–741.

Li,C. *et al.* (2010) Biomodels database: an enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Syst. Biol.*, **4**, 92.

Miller,A. *et al.* (2010) An overview of the cellml api and its implementation. *BMC Bioinformatics*, **11**, 178.

Miller,A. *et al.* (2011) Revision history aware repositories of computational models of biological systems. *BMC Bioinformatics.*, **12**, 22.

Rönnau,S. *et al.* (2005) Towards xml version control of office documents. In: *Proceedings of the 2005 ACM symposium on Document engineering*. ACM, New York, NY, USA, pp. 10–19.

Rönnau,S. *et al.* (2009) Efficient and reliable merging of xml documents. In: *Proceedings of the 18th ACM conference on Information and knowledge management (CIKM 09)*. ACM, New York, NY, pp. 2105–2106.

Rosado,A.L. *et al.* (2009) An XQuery-based version extension of an XML Native Database. In: *Proceedings of the 2009 EDBT/ICDT Workshops*. ACM, New York, NY, USA, pp. 99–106.

Saffrey,P. and Owen,R. (2009) Version control of pathway models using xml patches. *BMC Syst. Biol.*, **3**, 34.

Tyson,J.J. (1991) Modeling the cell division cycle: cdc2 and cyclin interactions. *Proc. Natl Acad. Sci. USA*, **88**, 7328–7332.

Waltemath,D. *et al.* (2011) Reproducible computational biology experiments with SED-ML—the simulation experiment description markup language. *BMC Syst. Biol.*, **5**, 198.

Wittig,U. *et al.* (2006) Sabio-rk: integration and curation of reaction kinetics data. In: *Data Integration in the Life Sciences*. Springer, Berlin Heidelberg, pp. 94–103.

Yu,T. *et al.* (2011) The physiome model repository 2. *Bioinformatics*, **27**, 743–744.