Master Thesis

semanticSBML

a Tool for Creating, Checking, Annotating and Merging of SBML Documents

Falko Krause
krause_f@molgen.mpg.de

February 7, 2008

Free University of Berlin
Department of Mathematics and Computer Science
Bioinformatics

Dr. Wolfram Liebermeister
Max Planck Institute for Molecular Genetics
Computational Systems Biology Group

Prof. Dr. Ulf Leser
Humboldt-University Berlin
Knowledge Management in Bioinformatics

Abstract

The *System Biology Markup Language* (SBML) is a common language for expressing biochemical sets of reactions that are accompanied by mathematical statements such as kinetic infomation. The program semanticSBML provides the systems biology community with the ability to integrate (*merge*) and *annotate* models with MIRIAM annotations. User interfaces are provided on multiple levels: application programming interface (API), console interface (CI), graphical user interface (GUI).

This work aims to enable a full support of SBML level 2 version 3 for the merging of models (including mathematical statements) and the manipulation of MIRIAM annotations (including annotation qualifiers). It is based on the previous work of the *Computational Systems Biology Group (Max Planck Institute for Molecular Genetics)* SBMLmerge. In its first development phase it extended SBMLmerge with a cross platform GUI and CI for all existing algorithms as well as a simplified API. In its second phase the underlying library (libSBML) was updated. The MIRIAM annotation manipulation as well as the merging algorithm was rewritten. The concept of annotation qualifiers was integrated. For the annotation and merging of models independent abstractions of systems biology models were developed. The merge abstraction is used for a better detection and resolution of conflicts in matching biological objects. Experiments were conducted to show the functional efficiency of the new algorithms as well as to show its possible uses.

Acknowledgment

I would like to thank Wolfram Liebermeister for his enthusiasm and for being the best tutor I could imagine. My girlfirend Jana for her patience and for giving birth to our child Nila. Edda Klipp and the Computational Systems Biology Group especially Jannis Uhlendorf, Anselm Helbig and Marvin Schulz for their work on semanticSBML (you created this too), Ulf Leser for sharing his independent view on our problems. The (lib)SBML community for driving me mad and helping me all at once. My family and friends.

A special thanks goes to Christian Ehrlich and Jonathan Schuld for proofreading my thesis.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 07. Februar 2008

Contents

List of Figures

1 Introduction

The field of systems biology tries to explain complex relationships in biological systems. Its focus is in the integration of information [1] to discover emergent properties that could not be revealed with other methods. A common approach in systems biology is to create a model of a metabolic process that has been researched by many scientific groups with the help of physical experiments. As the field of systems biology grows, so does the amount of information it generates - in particular the amount of models it has created. Models are expressed and published in various ways, from databases with custom datastructures to proposed exchange formats to verbal descriptions accompanied by mathematical statements.

The vision of systems biology is to not only study the behavior of selected biochemical networks but the biochemical network of a whole cell or even a whole organism. To reach this goal an integration of the generated models is needed. This integration can be achieved not only on a large scale e.g the integration of complete databases but also on a smaller scale for single models. Since systems biology is already strongly dependant on computational methods the preferred method of merging complex models is also a computer aided method. Two important preconditions enable this task: a common language for expression of models and a method for recognizing biological objects that describe a model (*biological entities*).

1.1 Preconditions

Language Formats There are a range of systems biology language formats. Each was created to express different aspects of a model. The most common ones are: CellMl [2], Proteomics Standards Initiative - Molecular Interaction XML [3] (PSI-MI), Biological Pathways Exchange Language [4] (BioPax) and Systems Biology Markup Language [5] (SBML). Besides many design differences [6] the main feature that differentiates SBML from the other language formats is that it supports mathematical statements to describe quantitative models. SBML is a widely used format (over 120 software tools support SBML [7]). SBML is a XML [8] derived and is developed in levels. Each level stands for an addition in the expressiveness of the format [9]. The current level is 2 which includes the creation of hierarchical models and usage of spacial characteristics. The language is specified in the "Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions" document [10] which is one of the main sources of this thesis (Appendix B provides a short overview of the most important concepts of SBML). SBML is becoming a (non official) standard for the exchange of mathematical simulatable models. An important feature of SBML is the availability of a programming library for its access and modification: *libSBML*.

Object Identification The subject of entity recognition was addressed by the development of the "Minimum information requested in the annotation of biochemical models" [11] (MIRIAM) rules. MIRIAM was developed as a team effort of systems biology scientists throughout the world and is hoped to

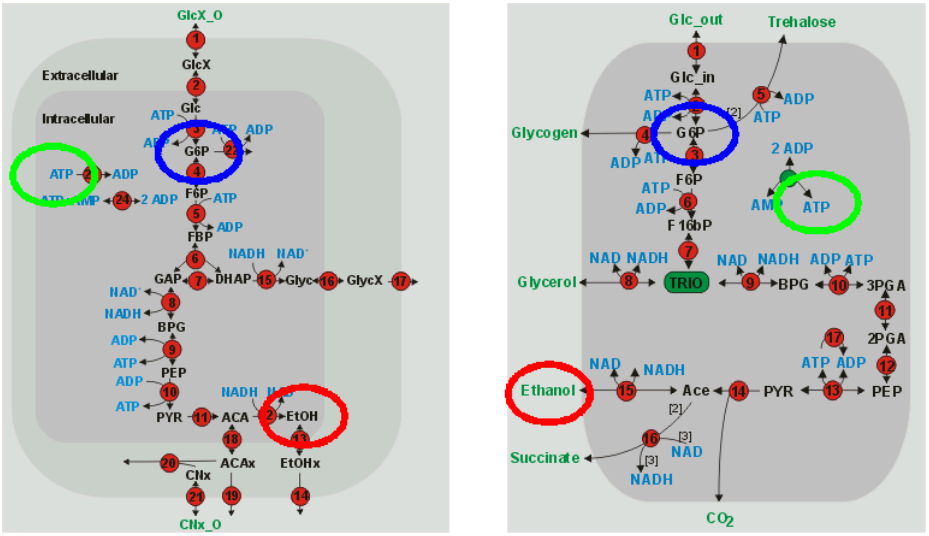

Figure 1: The figure shows two models of the glycolysis (left Hynne 2001, right Teusink 2000) that have common biological objects. Some objects can be easily recognized as duplicates (blue and green circles) others can not (red circles). MIRIAM annotations can resolve this situation by delivering a method for object identification.

standardize model curation. The goal of MIRIAM was to create a set of rules that ensure the quality of systems biology models. A part of the MIRIAM framework describes rules for the creation of a machine readable globally unique descriptions of biological entities. These rules were applied to the SBML format in the form of element annotations. In Section 3.2 the MIRIAM annotation in SBML will be introduced in detail.

Figure 1 shows an example of two models of the glycolysis, one from Tuesink *et al.* [12] and the other from Hynne *et al.* [13]. Even though both models describe almost the same aspects of the glycolysis, the recognition of duplicate objects can not be done by e.g., string comparison of entity names.

The article describing MIRIAM states : "We believe their [the MIRIAM rules] application will enable users to (i) have confidence that curated models are an accurate reflection of their associated reference descriptions, (ii) search collections of curated models with precision, (iii) quickly identify the biological phenomena that a given curated model or model constituent represents and (iv) facilitate model reuse and composition into large subcellular models." [1]. This thesis shows methods and examples that realize the expressed believes of the authors.

[1] "[...]" mark additions made by the author of this thesis

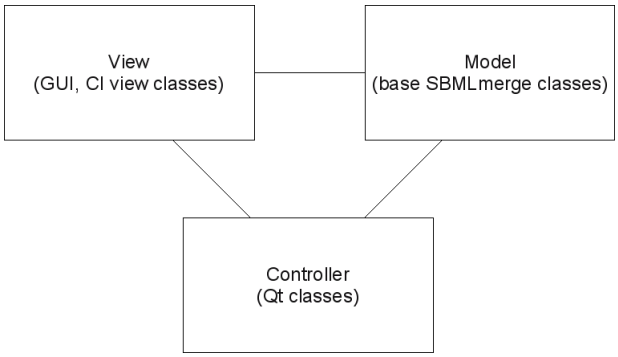

Figure 2: Model view controller pattern in semanticSBML.

1.2 Previous Work

In its previous work the *Computational Systems Biology Group (Max Planck Institute for Molecular Genetics)* developed the basis of semanticSBML, a program called SBMLmerge [14]. SBMLmerge is able to annotate SBML models with MIRIAM annotations which follow the first draft of the annotations format in SBML. Its merging algorithms are able to perform a successfull merging of two SBML models into one model. The resulting model has merged mathematical statements that were prooven to be functional for simulations. In addition to that it contains subrutines to perform a semantic check on a model and create a graphical representation of the model. The program is written in the Python programming language [15]. The algorithms of SBMLmerge follow the SBML format closely. Its design includes a console base user interface directly which is connected to the core algorithms. SBMLmerge has several shortcomings, some due to changes in the SBML format and others due to its design. Since SBMLmerge is the first tool of its kind it can be viewed as experimental software and many of its shortcomings are addressed in this work. SBMLmerge is currently the only publicly available tool that can manipulate MIRIAM annotations using a user interface and performed a merging of SBML models in a automated fashion. SBMLmerge is a generic tool that can be used offline (an internet connection is only needed during the installation). This philosophy is continued in semanticSBML. This thesis describes the development of semanticSBML that is based on SBMLmerge.

1.3 Procedure

Goals There were two goals for this masterthesis: on the one hand to create a fully functional version of the semanticSBML program based on SBMLmerge, that would include a graphical user interface (GUI) and a console user interface (CI) as well as a revised application programming interface (API). And on the other, the annotation and merge algorithm should be updated to fit the current status of the SBML and MIRIAM formats as well as eliminate flaws of the SBMLmerge algorithms.

User Interfaces To achieve these goals the existing code had to be adapted to provide interfaces that could be used as a direct programming interface and as an interface for the different user interfaces. In my research internship, that preceded this masterthesis, it was my task to design the basis for the graphical user interface. The SBMLmerge source code was currently being restructured. This lead to an interface based SBMLmerge core that could be used in a model-view-controller program design (see Figure 2). The GUI toolkit Qt [16] (developed by Trolltech) was chosen. It was the goal to create a program that could perform on multiple platforms. Until the end of my research internship the interfaces for the check, annotate and graph image generator subrutines were created. These interface subrutines were collected and wrapped (see Appendix A) into a single class that could be used as API. The class also stores the SBML model and is thus referred to as the *model* class. The model class was used as basis for the development of the user interface (*views*). In this thesis the model class and the user interface classes were extended to include the merging and model creation functions. In addition to the GUI a CI was created.

Official Release On the completion of the model classes and the views the cross platform ability was verified and the program was intensively tested. The project was then officially renamed to semanticSBML. The first step in the development of semanticSBML was concluded by an official release.

New Algorithms The first major change in the development branch was the portation (see Appendix A) to the current version of libSBML. In the current version the libSBML includes native support for the modification of MIRIAM annotations including annotation qualifiers. Qualifiers describe the relationship between two objects e.g., phosphate "is part of" ATP. Since the old MIRIAM annotation manipulation algorithm did not support qualifiers and a new method of editing a MIRIAM annotations was available, the algorithm was rewritten. For a merging of SBML models MIRIAM annotations are essential as previously mentioned before. Changing the annotation interface also meant changing the merging algorithm. Furthermore the merging algorithm of SBMLmerge was strongly geared to the SBML format and disregarded the biological context of the model. This lead to a complete redesign of the merge algorithm.

Some features of the new algorithm are: a more biologically meaningfull merging of models (e.g., location of physical biological entities is respected: ATP in cytosol is not the same as ATP in the mitochondrion), merging of multiple models at once (in SBMLmerge only pairwise merging is possible), overview and the controll over which entities should be merged and that mathematical statements are connected to the entities they describe. The new merging algorithm works on the basis of semanticSBMLs own abstraction for systems biology models (the core abstraction is shown in Figure 3).

Merge Concept The semanticSBML abstraction is based on the following concept: a systems biology model consists of multiple *biological entity*s. Biological entities can be identified (by MIRIAM annotations). Each entity has a *biological quantity* that describes the entity (e.g., unit, location, quantity type). A *model* makes *statement*s about biological entities (e.g., the mathematical statement

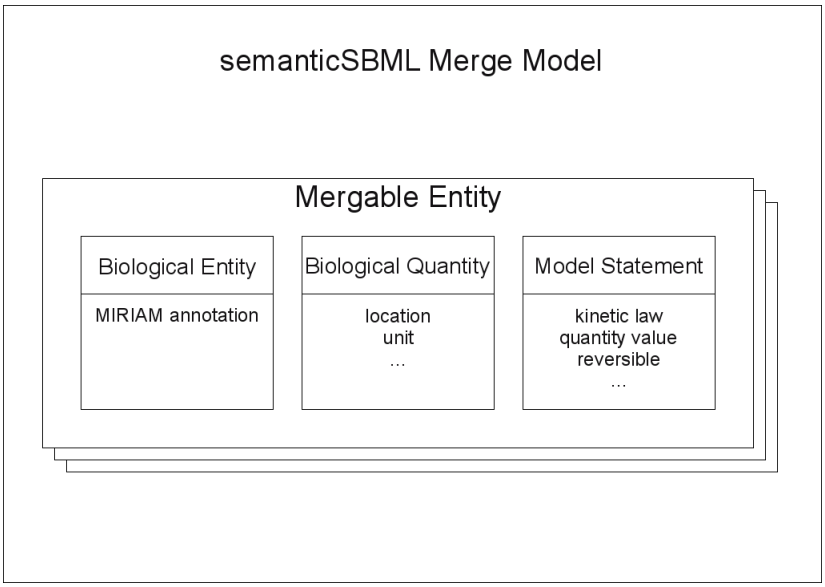

Figure 3: In semanticSBML a systems biology model consists of many mergable entities. The mergable entity consist of a biological entity a biological quantity and a model statement. Examples of the contents of each of these objects are shown.

of a kinetic law, the value of the amount of an physical biological entity, the reversibility of a reaction).

To merge multiple models, duplicate entities are identified and a list of duplicate entities is generated (*merge tuples*). The list of merge tuples can be manipulated by a user since the duplicate detection may not be correct or may not reflect the wishes of the user. From the merge tuple a new entity can be created. However the merge tuple may contain conflicts since e.g., different models may make different statements about the duplicate entities (e.g., the entities have different initial concentrations). The user must resolve all conflicts (e.g., choose the correct initial concentration). After the resolution of all conflicts, a merged model can be created. The resolution of conflicts and the creation of the final model is associated with many difficulties. An example for such a difficulty would be the fact that a SBML model must contain unique identifiers for each element. A detailed description of the merge concept can be found in Section 3.3.

1.4 Experiments

Databases There are a number of databases that support the export of SBML models: BioCyc [17], Reactome [18], JWS online [19]. These databases however do not support MIRIAM annotations in their current version. The largest source of curated models, which includes MIRIAM annotations, is the BioModels [20] database. It contains models from diverse sources that are added to the database after a curation step in which among other things the MIRIAM

annotations are added. This makes it the best source for real life examples of models that use the SBML format. The BioModels database was used not only in the experiment as a source for models but also during the whole development of semanticSBML.

Experiments To exemplify the functional efficiency and the potential uses of the new algorithms three experiments were conducted. In the first experiment a clustering of the BioModels database was performed to exemplify a fully automated method for finding similar models with the aide of the MIRIAM annotation API. In the second experiment the merging of the previously introduced glycolysis models using semanticSBMLs new merge algorithm was attempted. The third experiment consisted of the merging of a single celled model of an "... autonomous metabolic oscillations in continuous culture of Saccharomyces cerevisiae" by Wolf *et al.* [21] into a two celled model.

1.5 Organization of this Document

Special terms that a reader of this thesis should be aware of can be found in the Appendix A. An overview of the most improtant SBML elements can be found in the Appendix B. SBML elements play an important role in Sections 3.2, 3.3 and 4. Function and variable names are written in `typewriter font`.

This thesis describes each of the development steps in detail. Whereas Section 2 has its main focus on the implementation of a fully functional release of semanticSBML. Section 3 has its main focus on the concepts of the new annotation and merge algorithm with less focus on the implementational details. In Section 4 the experiments will be presented. The conclusion in Section 5 is followed by and overview of planed enhancements of semanticSBML in Section 6.

2 Phase I

In the first phase the goal was to create a fully functional release of semanticSBML. The release includes the library API as well as a graphical and console user interface. Since parts of the GUI already existed the GUI had to be extended to include the missing merging, annotation and model creation functions (see Appendix A). For a public release clean-up work and updating of user manual files had do be done. In addition to that the program was adapted to use the latest version of the Qt library.

The first official release of semanticSBML contains the following functions.

Short Name	**Description**
display	Create a graph visualization of a SBML model.
id to SBML	Create a model from a list of database identifiers.
check	Execute a semantic check on a model, e.g., check for missing MIRIAM annotations.
annotate	Search, add and remove MIRIAM annotations for SBML model elements.
merge	Integration of multiple SBML models into one model.

After the release the source code was branched into two versions: a stable release version and a development version. The release version was kept for patches of major program flaws (one patched release was issued) while the main development continued in the branch. The new development will be described in Section 3.

The GUI and CI are currently used as a template for the creation of a web interface which is currently under development by a colleague.

2.1 Porting to Qt4

Preliminary Work The creation of the GUI was started during my research internship. The development described in this thesis picked up the work where the research internship left off. It was decided during the research internship that Qt should be used as a widget (see Appendix A) toolkit. Qt provides a stable cross platform library for the creation of GUIs. Qt is written for the C++ programming language. For Python an interface is provided by the PyQt [22] (developed by Riverbank) library.

Porting Mandatory At the beginning of the masterthesis the support of Qt version 3, the current version at the time of my research internship, was discontinued due to the release of Qt version 4. The new version contained non backwards compatible changes in the application programming interface (e.g., functions were renamed) and thus required the porting of the existing source code. The final decision to port to Qt4 was made when the official distribution webpage of Qt no longer offer binary installation files for Qt3.

Procedure Since this project is based on the PyQt and not directly on the Qt library the porting tools that were provided by Trolltech could not be used. Instead a series of regular expressions were written, that could partly be applied

without human interaction. However in most cases find and replace operations had to be applied by hand. All regular expressions that were used were collected and added as a resource to the project. This resource was also used by my colleague for the porting of the web interface. Publishing this resource will mostlikely help other developers porting their GUI from PyQt3 to PyQt4.

```
1  #directly applyable regular expressions
2  sed -i 's/QObject/QtCore.QObject/' $@
3  sed -i 's/SLOT/QtCore.SLOT/' $@
4  sed -i 's/QString/QtCore.QString/' $@
5  sed -i 's/QGridLayout/QtGui.QGridLayout/' $@

   ...

36 sed -i 's/QDialog/QtGui.QDialog/' $@
37 sed -i 's/QTabWidget/QtGui.QTabWidget/' $@
38 #remove extras, form previous porting attempts
39 sed -i 's/QtGui.QtGui/QtGui/' $@
40 sed -i 's/QtCore.QtCore/QtCore/' $@
41 #replace import
42 sed -i 's/from qt import ./from PyQt4 import QtCore,QtGui/' $@

44 #api differences within classes
45 #-------------does not exist any more
46 #qApp.setMainWidget(gui)
47 #setMultiLinesEnabled

49 #replace manually, regular expression may not be exact
50 's/insertTab(\(\S*,.*\),\(.*\))/insertTab(\2,\1)/'
51 #WARNING unsafe this will mess up the previous changes
52 's/insertTab(\(\S*,.*\))/addTab(\1)/'
53 's/\.message(\(.*\))/.showMessage(\1,2001)/'
54 's/setCurrentPage(.*indexOf(\(.*\)))/setCurrentWidget(\1)/'
55 's/setCurrentPage(\(\d*\))/setCurrentIndex(\1)/'

57 #incomplete, this depends on programing style
58 's/qApp.processEvents/QtGui.qApp.processEvents/'
59 's/\.setMultiSelection(1)/.setSelectionMode(QtGui.QAbstractItemView.MultiSelection)/'
60 's/selectionChanged/itemSelectionChanged/'
61 's/QWidget(\(.*\),.*)/QWidget(\1)/'
62 's/QGridLayout(\([^,]*\),.*)/QGridLayout(\1)/'
63 's/QTabWidget(\([^,]*\),.*)/QTabWidget(\1)/'
64 's/QLineEdit(\([^,]*\),.*)/QLineEdit(\1)/'

   ...

81 's/QWidget.close(\(.*\),.*)/QWidget.close(\1)/'
82 #emit does not take tuples anymore
83 's/emit(\([^,]*\),\s*(\(.*\)))/emit(\1,\2)/'
84 's/setCaption(\(.*\))/setWindowTitle(\1)/'
85 's/QScrollArea(\([^,]*\),.*)/QScrollArea(\1)/'
86 's/addChild/setWidget/'
87 #change treeview items selection state
88 's/\S*setSelected(\([^,]*\),\(.*\))/\1.setSelected(\2)/'
89 's/insertItem(\(.*\))/insertItem(0,\1)/'
90 's/Qt.Align\([^)]*\)/QtCore.Qt.Align\1/'
91 's/QFrame(\([^,]*\),.*)/QFrame(\1)'
```

The list is shortened since it is only used to show examples, "..." stands for excluded text. The regular expressions in the listing above are ordered from directly applyable regular expressions to those that have to be checked carefully by hand. The directly applyable regular expressions are written so that they can be copied into a file and used as a shell script that calls `sed` (stream editor for filtering and transforming text). All other regular expressions can be copied into `vim` (Vi IMproved, a programmers text editor) and applied if a correct match is found. The regular expressions of this section depend on programming style.

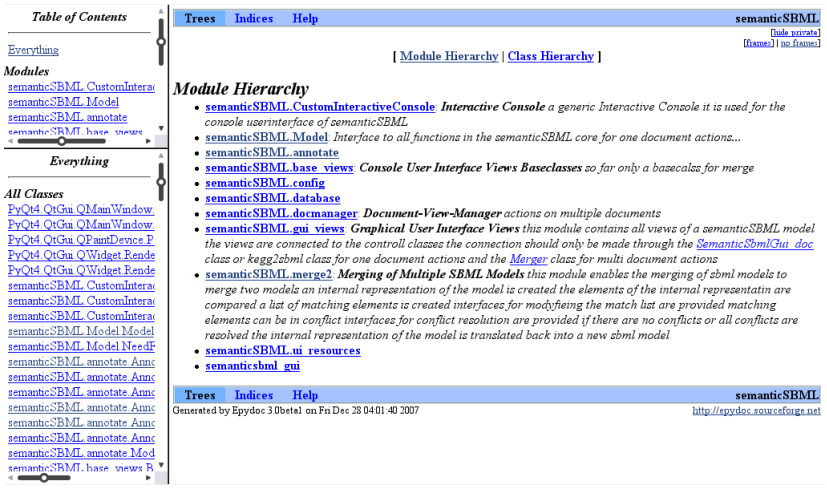

Figure 4: Source code documentation with Epydoc.

2.2 Application Programming Interface (API)

Documentation The Python programming language provides native methods for source code documentation. These native methods can be extended by external tools to generate a clearly structured source code documentation that includes hyperlinks between classes, text highlighting and input/output type description among other things. The description of input and output of functions in Python is especially important since Python uses a dynamic type system (usually referred to as duck-typing - types of variables are assigned dynamically by detection of the type of the first inserted value of the variable). The dynamic type system requires a proper source code documentation. Without the documentation a user of the semanticSBML API would have to guess input and output types. For this purpose Epydoc [23] (Automatic API Documentation Generation for Python) was chosen (see Figure 4). All functions of semanticSBML that belong to the library interface are documented in detail including input and output parameters.

Implementation An over view of the API class structure can be seen in Figure 5. The development of the user interfaces required an interface class (model class). The model class was partly developed by my colleague during the restructuring of SBMLmerge and was refined by myself in the creation of semanticSBML. At the same time I developed a module (see Appendix A) that provided the abstraction of an SBML models for the user interfaces. The general interface to SBMLmerge is provided by the class `Model`. The following list shows all important functions of the model API.

Function Name
Description

`__new__`
Initialize the model from either a string of the filename, another `Model` class instance, a libSBML `document`, `None`: create a new model.

`save_as`
: Save model under the specified filename using the `save` function.

`save`
: Save the model as SBML file on the harddisk.

`get_libsbml_document`
: Return a libSBML document instance.

`get_libsbml_model`
: Return a libSBML model instance.

`get_model_as_svg`
: Return a graphical representation of the model in the Scalable Vector Graphics (SVG) format as string. This function requires the program `dot` (Graphviz).

`check`
: Run a semantic check on the model and return a check-results object. The semantic check is a function of SBMLmerge and is documented in with SBMLmerge.

`addAnnotationLink`
: Add an annotation link (Database,Identifier) to a specified element if the annotations is incorrect an `InvalidAnnotationError` is raised.

`addAnnotationLinkAutomatic`
: Add MIRIAM annotations automatically - if no annotations were found an `InvalidAnnotationError` is raised.

`delAnnotationLink`
: Delete an annotation link, see `addAnnotationLink`.

`getNumNotAnnotatedElements`
: Return the number of elements that are not MIRIAM annotated.

`getAnnotationElements`
: Return a list of annotation elements of this model.

`getAnnotationSuggestions`
: Return a list of suggested annotations for an inserted libSBML element.

`getAnnotationQuerys`
: Return a list of query strings that can be used to search MIRIAM annotations in the internal database for an inserted libSBML element.

`getAnnotation`
: Return the MIRIAM annotations as a list of database, identifier tuples of the inserted libSBML element.

`issetAnnotation`
: Return `True` if the element is MIRIAM annotated. This function should not be used but rather `getAnnotationStatus`.

`getAnnotationStatus`
: Get the status of the MIRIAM annotation of the inserted libSBML element.

`id2str`
: Return a human readable string representation of an inserted database,id tuple.

`id2str_compartment`
: Return a human readable string representation for a libSBML compartment element.

`id2str_reaction`
: Return a human readable string representation for a libSBML reaction element.

`compartmentIds`
: Return a list of all libSBML compartment ids.

`databases`
: Return the list of databases used in the internal database for an inserted libSBML element. If no input: return all databases.

It is important to note that the model class only includes functions that are working on a single model. The merge functions work on multiple models and thus are found in a different class.

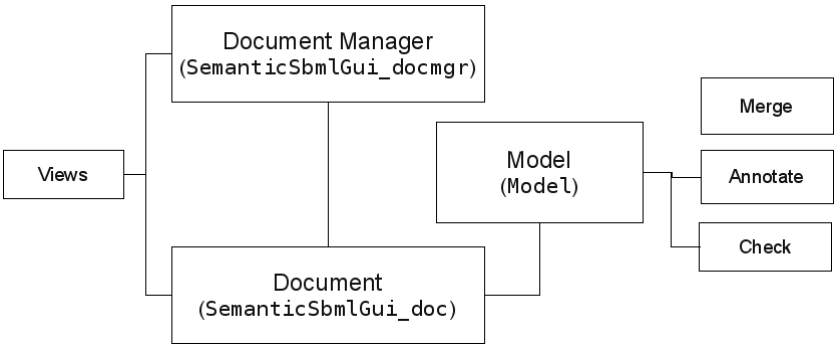

Figure 5: semanticSBML API: The Model class delivers a general programming interface, which is extended by the Document Manager and the Document class for the GUI and CI views.

Specialization of the User Interfaces The user interface module is located in the `docmanager` module. The modules extends the `Model` class with file management functionalities. It contains two classes `SemanticSbmlGui_docmgr` (*document manager*) and `SemanticSbmlGui_doc` (*document*). The document class represents a single model and is the direct extension of the model class. In additional to the model class functions it provides state variables. The state variables are set in the wrapped model functions. If a state is modified a signal

(see Appendix A) is send so that all views depending on this document can execute necessary functions for e.g., updating the view. The document manager class manages multiple documents. It creates and connects views to the documents. To enable one document and multiple document actions in a user interface the document manager provides functions to show the number of active (selected) documents and to change the activation state of multiple documents. It also takes care of the safe closing of documents to avoid data loss.

2.3 Graphical User Interface (GUI)

The GUI is provided by a main class which creates the main window. In the main window the user can load documents and a list of loaded documents can be seen. By selection of a document item in the main view, documents can be activated. An activation or deactivation will enable or disable the push-buttons that trigger the creation of views. Each of the functions of the core code are represented by a view. A view is created in a new tab of the main tab widget. The tab has the same name as the function key and may contain another tab widget. In the following the views for the merging and creation of models algorithms will be described.

2.3.1 Model Creation

Interface Usage In semanticSBML models can be created by the insertion of a list of KEGG [24] reaction identifiers. The GUI view provides this functionality in a two step process (see Figure 6). The first step is an insertion of list of KEGG reaction identifiers. This can be done either by file or directly. On the inserted text (from the input form or from the file) a regular expression is applied that filters all KEGG reaction identifiers. This means that another SBML file can be used as input. In the second step all reactions are presented in a human readable form. The list can then be modified by using the "back" push-button. This will show the first view again with the filtered list of identifiers in the input textfield widget. To create a proper model the compartments of the reactions can be specified. From a list of Gene Ontology [25] identifiers one entry can be selected with the help of a dropdown box widget. The creation may fail. In this case an exception is raised by the creation class that will be presented in a error message box. If the creation was successfull the view class will send a signal that a new document was created. This will trigger a function that adds the new document to the main view. The new document can now be treated like any other externally created SBML model.

Implementation Even though a view in a strict sense only represents existing models, a model creation view was created (see Figure 7). It contains three important functions: `makeInitMenu`, `slotNext`, `slotKegg2Sbml`. In creation of the class instance the `makeInitMenu` is called. This creates the first view. Two slots can be called from that view. The first opens a file-open pop-up widget (`slotInfileBrowse`) and the second creates the second view (`slotNext`). The file input folder of the last file opening is stored over multiple session with the help of the `Config` class. In the second view functions of the API (see Section 2.2) are used to display humand readable representations of the reactions us-

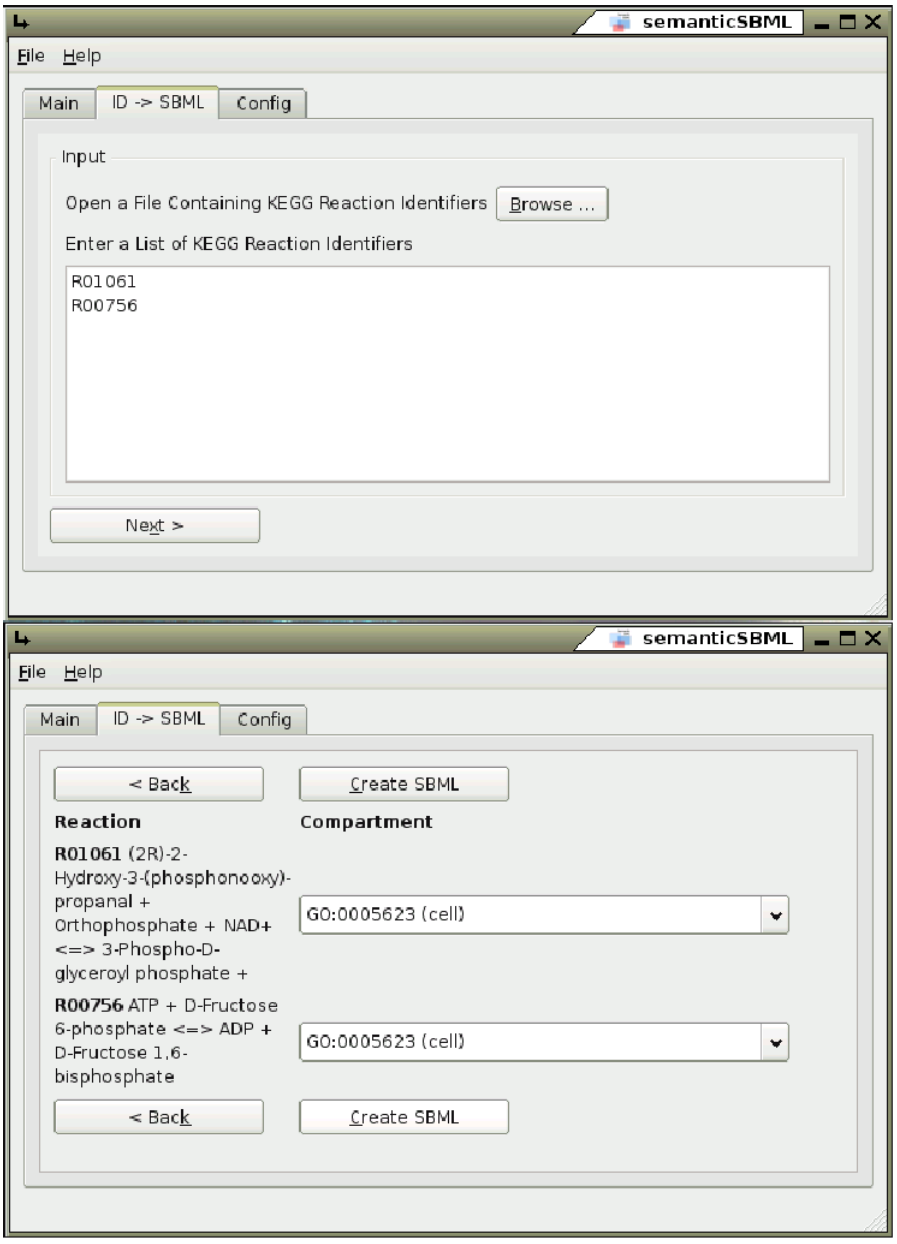

Figure 6: The model creation view enables the creation of models in two steps. First step: input of KEGG reaction identifiers. Second Step: visual representation of reactions and choice of compartments (Gene Ontology identifier)

ing the `id2str` function and displays a list of Gene Ontology identifiers using the `compartmentIds` and `id2str_compartment` functions. By using the "Create SBML" push-button, the slot `slotKegg2Sbml` is called. It will invoke the `kegg2sbml` function of the `kegg2sbml` module. If a `KEGG2sbmlError` is returned

Figure 7: Create Model view class functions.

by the `kegg2sbml` function, the creation failed and the error message is displayed in a pop-up window.

2.3.2 Merge

Interface Useage If a user wants to merge models in the semanticSBML GUI he has to select the desired models in the main view and press the merge push-button. It is only enabled if more than two models are selected. Upon the pressing the merge button the modification state of the models is checked. If a model is unsaved the merging is aborted and a error pop-up appears warning the user to save the model first (documents states see Section 2.2). If the models are not modified there are two possible results. The first result can be that the merging was successfull without user interaction. The merged model is then returned right away. The new model will appear in the main window having a modified state. The second result is that a new tab appears that again contains a tab widget in which the merging takes place (see Figure 8). The merging is executed on models pairwise. If the models that are to be merged contain duplicate entities that have conflicting values, the attributes of the each entity will be displayed in a vertical list. In between the two entities the merged attribute values are displayed. If the values are in conflict a conflict resolution widget is displayed (see Figure 8). There are two possible conflict resultion widgets. A modifiable dropdown box and a non modifiable dropdown box. Conflicting values of e.g., type `bool` can not be modified in comparison to floatingpoint numbers. The user has the choice to resolve the problem by using the values from the resolution widgets or by using the values of either of the elements for this one entity. In addition to that the user can chose to solve every conflict by using one of the models values for all conflicting entities. There is also a option to keep both entities however it is disabled by default and has to be enabled in the configuration tab. Choosing to keep both entities may result in a erroneous model. After resolving all conflicts in duplicate entities it is checked if circular rule definitions exist, rules that are defined by itself (directly or indirectly). To resolve this a graphic of the circular rule and its alternatives is drawn and alternatives can be chosen with push-buttons (see Figure 9).

Implementation In its first development stage the merging of models in semanticSBML uses the SBMLmerge algorithms. These algorithms were modified in order to create an interface based merging algorithm that could be used in a model-view-controller software design pattern. The SBMLmerge merge algo-

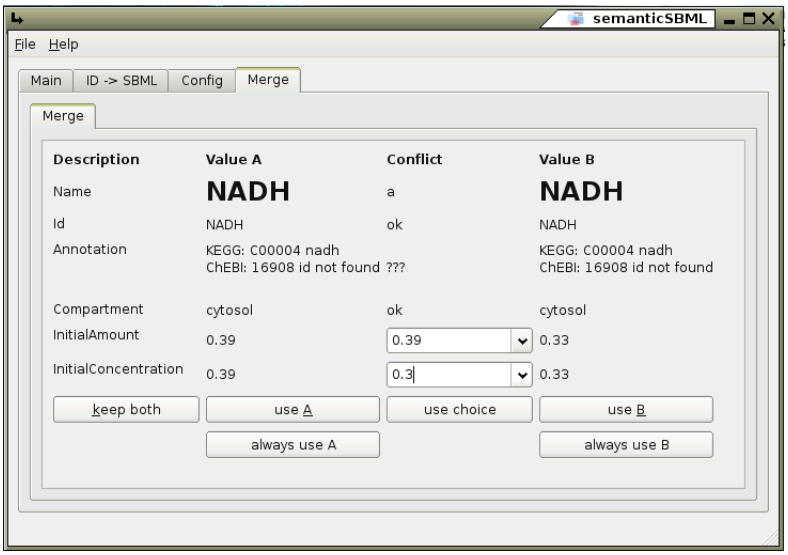

Figure 8: Conflict resolution in the old semanticSBML GUI based on SBMLmerge.

rithm is designed to merge models parwise. To enable a merging of an arbitrary number of models the merge algorithms is first called with two models and then subsequently with the resulting (merged) model and one of the remaining models. The interface to the merge algorithm was developed in cooperation with my colleague to contain the following functions.

Function
: Description

`__init__`
: Initialize the merge class with two models as input.

`find_collision`
: Find duplicate entities that contain conflicting values.

`resolve_collision`
: Resolve the conflict of the duplicate entities values.

`find_circle`
: Find circular rule definitions and also resolve the circular rule definition problem by adding extra parameters.

`deleteunused`
: Delete unused elements in the created model.

`finish`
: Return the newly created model.

To merge two models the `Merger` class is initialized with two model instances. The user cannot see or influence the compare and initial matching process. A

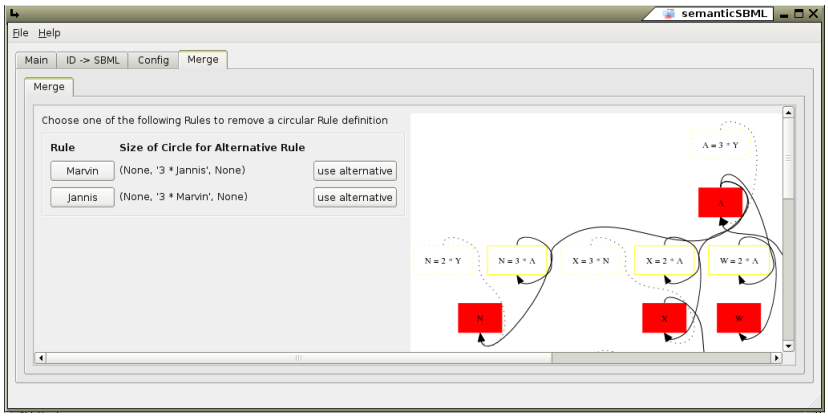

Figure 9: Resolution of circular rule definitions.

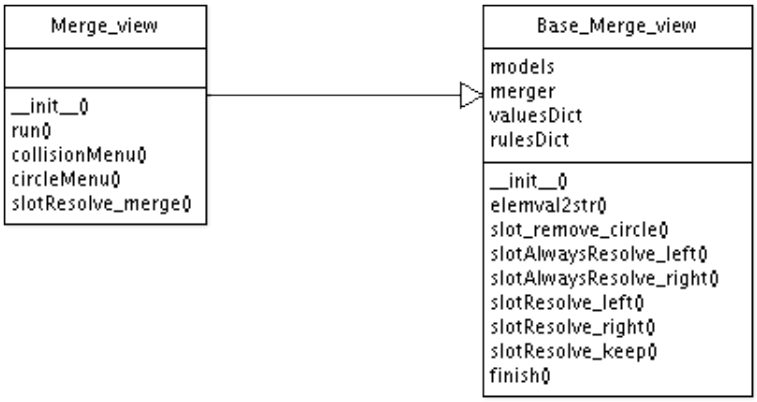

Figure 10: View classes of the merge GUI based on the SBMLmerge algorithm.

list of duplicate entities can not be accessed through the interface. In the initialization the function `find_collision` is called. If it returns `None` there are no duplicate entities that have conflicting values. If the function `find_collision` returns an object duplicate, entities that contain conflicting values were found. The returned object contains the duplicate libSBML elements as well as a dictionary (see Appendix A) of the values of the entities with flags that show if the values are in conflict. The values are then ordered by their biological importance and converted to string representations (`elemval2str`) if necessary. The dictionary keys contain descriptions of the values. This information is used to create the user interface widgets. When the user resolves the conflict by pressing one of the push-buttons a "resolve" function is called. The function `resolve_collision` has multiple conflict resolution strategies (e.g., use values of the left element, use the values chosen by the user). Each resolution strategy demands a specific input. Since the user interfaces all have a similar design the base class `Base_Merge_view` was created (see Figure 10) to unite the input generation for the `resolve_collision` function. The view class resolve function can return string representations. One of the tasks of the base class is the recon-

version of these string representation. The call of the function `find_collision` is repeated with the steps just described until no more duplicate entities with conflicting values are found.

Experience showed that circular rule definitions cannot be found while merging two man made models, nevertheless the `SBMLmerge` algorithms can detect and resolve this problem. The function `find_circle` returns a list of rule definition identifiers and their mathematical statement that have a circular definition. Alternatives to each of the rules are given. Furthermore a graphical representation of the rule dependencies is returned that should aid the user choosing the correct rule. The rule identifiers and their mathematical statements are displayed in a table with push-buttons that show the rule identifier. Next to it the graph representations of the dependencies is displayed. Similar to the conflict resolution the resolution of circular rules is repeated until all problems are resolved. Some choices may lead to loops in the resolution.

Discussion A compromise had to be made between the functional efficiency and the implementational effort. The conflict resolution algorithm has a number of problems that limits its usability. Some of the limits are its pairwise approach in merging, the inability to handle entities that seem identical but that are marked as non identical, its inability to recognize biological facts (like the location of a physical entity) and the fact that most of the merging process is hidden for the user. These limits will be addressed in the new merging algorithm that is presented in Section 3.3. The circular rule definition algorithm and its user interface is very hard to understand by a user not familiar with concrete algorithms. Since it is one of the features of SBMLmerge it was integrated into the user interface. For an actual use it needs further improvements. The implementation of the user interface for this algorithms helped to understand the functions of SBMLmerge in detail and to analyze its shortcomings.

2.4 Console Interface (CI)

The console user interface provides a user interface that is not dependant on a X Window System (however it is still dependant on Qt due to signals send from the document manager). One of the main features of the CI is its batch processing ability. The desired audience of semanticSBML are system biology scientists that create and simulate models in a mostly tool driven process. The CI provides an interface that can be easily automated without knowledge of the Python programming language.

The console interface is geared to the interactive Python console. It was developed since no similar Python module was available. It is designed to aid users with little programming knowledge (in Python or in general) creating simple automated tasks with semanticSBML.

Implementation - Concept The console interface functionality is provided by the `CustomInteractiveConsole` module (see Figure 11). It consist of two classes: the main class `CustomConsole` and a singleton datastorage class `Singleton_datastore`. The main class can be instantiated with a dictionary containing commands as keys (strings) and function pointers as values. When

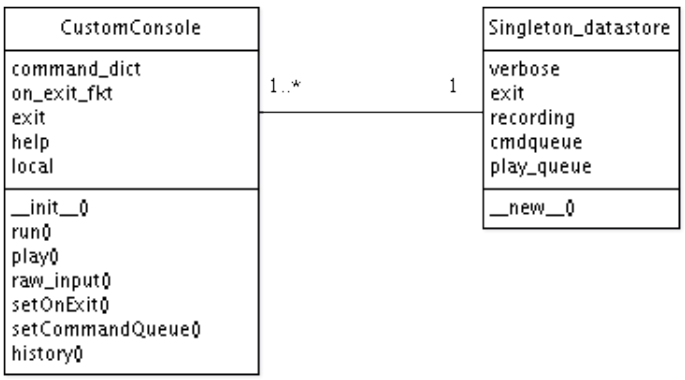

Figure 11: The console interface is based on the `CustomInteractiveConsole` module. The module consists of two classes: the `CustomConsole` and the `Singleton_datastore`. The `Singletone_datastore` provides a singleton class that stores values that can be used in nested instances or that should be applied on all existing instances of `CustomConsole`.

the `run` function is called a loop is started that prompts the user for an input and executes the according function if the input matches a keyword from the command dictionary. If the user did not enter a keyword all available commands are displayed. Since sub menus are needed e.g., main-menu → annotate-menu it is possible to open a new `CustomConsole` instance within a running loop. The function `raw_input` provides the ability that only single values can be retrieved. This function is also used internally to retrieve user input.

Implementation - Batch Processing To enable a batch processing ability a singleton class was created. Since multiple independent instance of the main class can be created (sub menus, single value user input) the singleton class provides the ability to share a global queue of commands. The singleton class contains two important variables: `cmdqueue` and `play_cmdqueue`. The `cmdqueue` contains a list of strings (keywords). If the batch processing is activated the list of commands (`cmdqueue`) is copied to the `play_cmdqueue`. Each time the `raw_input` function is called the function checks if there is an active command queue (`play_cmdqueue`). If this is the case a value is taken from the `play_cmdqueue` and returned until all commands on the queue were executed. The batch processing mode can be activated on the instantiation of the main class or within a running loop. Commands can be added at instantiation or recorded during an interactive session. On setting the command queue it is serialized (see Appendix A). It is then deserialized during the instantiation of the singleton class. Similar to the command queue a history of commands is kept that can be copied to the `play_cmdqueue`.

Implementation - Safety There are two methods to exit a console interface session: a local exit (for a single instances) and a global exit (for all nested instances). In some cases an on-exit-function is needed for e.g., the safe closingof documents. The on-exit-function can be set for each instance of the main

class by using `setOnExit`. This function is called during each exit attempt and may prevent the exit.

Implementation - Integration In the following example the module is loaded and a dictionary of commands and function as well as a help text is created and inserted into the `CustomConsole` class. The command loop is then started with the `run` function.

```
from semanticSBML.CustomInteractiveConsole import CustomConsole
self._locals  = {
       'l':(self.listFiles,'List Models'),
       'i2s':(Id2Sbml_view,'ID -> SBML'),
       'd':(self.openDirectory,'Open Directory')
       }
self._help="
<<< semanticSBML main menu >>>
l             list all loaded models
d <DIR> open all models in the directory (without arguments last used)
i2s           ID -> SBML Generate SBML files from Database Identifiers"

cc = CustomConsole(self._locals,self._help).run()
```

By invoking the creation of the `Id2Sbml_view` class a submenu is created. The source code of the class is shown in the next example.

```
class Id2Sbml_view(QWidget):
    def __init__(self):
        help="<<< ID -> SBML >>>
e <ID1 ID2>    Enter a List of KEGG Reaction Identifiers
q              exit this menu"
       cc = CustomConsole({'e':(self.slotNext_l,'insert list'),'q':(self.exit,'exit')},...
         ...help).run('...')
```

Also user input can be returned directly without connecting it to a function.

```
input = CustomConsole().raw_input('Are you sure you want to do this? y/n:')
```

The `raw_input` function of the `CustomConsole` class is used instead of the native Python `raw_input` function since its input is captured and can be replayed. Just like the GUI, the CI interface consists of a main class and views for each of the functions of semanticSBML. The views can be found in the `console_views` module and correspond in their make-up to the GUI views.

Interface Usage

```
$ ./semanticsbml_console.py -h
usage: semanticsbml_console.py [options] <sbml .xml files>

options:
 -h, --help     show this help message and exit
 -q CQ, --cmdqueue=CQ set the command queue
 -v, --verbose   let functions crash
 -p, --play     play stored command queue on start
$ ./semanticsbml_console.py -q 'c 0;a 0' -p ../../../release_25September2007_sbmls/...
  ...curated/BIOMD0000000090.xml

[ loading identifier database... 49143 entries loaded in 0:00:03.197326 ]
[ loading reaction database... finished in 0:00:05.092594 ]

opened BIOMD0000000090

    <<<semanticSBML main menu >>>
l       list all loaded models
o <FILENAME> open a model
d <DIR>    open all models in the directory (without arguments last used)
c <MODEL_NR> display check results for a model
a <MODEL_NR> annotate a model with database identifiers
```

```
e       export a svg image of a model
m [<MODEL_NR1> , ..] merge 2 or more models by inserting a list of models
s       save model(s)
v       save model(s) as
r <MODEL_NR> remove model
about     about this software
i       ID ->SBML Generate SBML files from Database Identifiers
commands: help, dir, rec, prec, play, hist, q, exit
you can use ctrl+D (win ctr+Z) to exit
###executing### c 0
->semantic check (0/0/0)
->annotation check (0/10/1)
    warning: Reaction v2: no Annotation recognized
    warning: Reaction v10: no Annotation recognized
    warning: Reaction v14: no Annotation recognized
    warning: Reaction v4: no Annotation recognized
    warning: Reaction v5: no Annotation recognized
    warning: Reaction v6: no Annotation recognized
    warning: Reaction v7: no Annotation recognized
    warning: Reaction v15: no Annotation recognized
    warning: Reaction v17: no Annotation recognized
    warning: Reaction v18: no Annotation recognized
    information:Model contains elements with missing or unrecognized annotations. ...
     ...Currently only the 'is' qualifier is recognized, all annotations with other ...
     ...qualifiers are not recognized. Please annotate your model, or wait for a new ...
     ...version of semanticSBML. SBMLcheck depends on annotations

->semantic dependency check (3/1/0)
    error: According to their annotations the reactions v8 and v9 are identical
    error: According to their annotations the species S1 and S2 are identical
    error: According to their annotations the species C1 and C2 are identical
    warning: 10 reactions could not be checked due to missing annotations
->conservation constraint check (0/1/0)
    warning: 21 reactions not checked for conservation relations due to missing ...
     ...annotations
->overlap check (0/0/0)
->physical value check (0/0/0)
->rules check (0/0/0)
###executing### a 0
0 Model BIOMD0000000090
1 List of Compartments
2   .external [ID:c0](annotation not supported)
3   .cytosol [ID:c1](annotation not supported)
4   .mitochondria [ID:c2](annotation not supported)
5 List of Species
6   SO4_ex [ID:sul_ex]
7   EtOH_ex [ID:eth_ex]

...

31   Hm [ID:Hm]
32 List of Reactions
33  .v1 [ID:v1](annotation not supported)
34  .v13 [ID:v13](annotation not supported)
35   v2 [ID:v2](bad annotation)

...

52  .vLEAK [ID:vLEAK](annotation not supported)
53  .v12 [ID:v12](annotation not supported)

    <<<Annotation Menu>>>
l       list elements (without annotations)
la      list elements and their annotations
d <ELEMENT_NUM> delete annotation
a <ELEMENT_NUM> add suggested annotation/ automatically annotate "List of .." or whole ...
 ...Model
s <ELEMENT_NUM> <QUERY> search and add identifier
f <ELEMENT_NUM> <DB> <ID> add an identifier directly (DB: KEGG, GO ...)
q       back
commands: help, dir, rec, prec, play, hist, q, exit
you can use ctrl+D (win ctr+Z) to exit
...exit
closing all documents now
$
```

The example above shows an interactive CI session. The CI is called with a list of commands that are then processed (line 9). The session starts by setting the switches to open a file and setting the command queue to the commands `c 0` and `a 0`. The `-p` switch executes the command on startup after opening the file. The command `c 0` executes a semantic check of the open model number `0` (line 31) and is followed by the calling of the annotation view with the command `a 0` (line 56). The view diplays the complete annotations status of the model (lines 58 to 109) followed by the available commands that can be used to manipulate the annotations (lines 112 to 118). The call of the annotation view created a nested instance. Both instances are exited by using the `exit` command (line 121). The string "closing all documents now" indicates that the modification status of all open models is checked.

Discussion The console class is a generic class. It is my hope that it will be reused by other developers since it is distributed under the same open licence as semanticSBML.

2.5 Beta Release

General There are two methods to install semanticSBML. The first method is the source installation, and the second is a packaged installation. The installation method depends on the operating system. The creation of the distribution packages will be described in the Sections 2.5.1 and 2.5.2. For the beta release the `INSTALL` file was updated to describe the installation process in detail. The `READEME` was updated to contain a basic description of the current functions of the program.

Clean-Up To prepare the official release the root folder of the project had to be cleaned up from test scripts and library scrips in development. Since semanticSBML is a project that includes source code that has been developed by my colleagues and myself, moving and removing of files had to be done carefully and with the agreement of the corresponding developer. The inclusion criteria of executable scripts was that they had to at lease return a list of switches.

Publication To complete the release the Sourceforge project site was updated and screenshots of the program were uploaded. The project was renamed to semanticSBML and the distribution packages were uploaded. After the update of the institute homepage of the project, an announcement was made to the libSBML mailing list.

2.5.1 Source Installation

To use semanticSBML on the Windows (Microsoft Inc.) or *OS X* (Apple Inc.) platform, it has to be installed with the Python installation script (from the package Python Distuils).

Installation Difficulties The installation requires that all dependencies on external libraries (libSBML, Qt, PyQt, Graphviz, SOAPpy) are satisfied before the installation is started. All versions semanticSBML up to 0.9.3 are dependant on libSBML 2.4.* this is again dependant on Python 2.4.* (libSBML 2.4.*

will not work with newer versions of Python). Installing Python and libSBML is straightforward since both exist as binary distributions for Windows and *OS X*. The dependencies on PyQt4 and Qt4 however is problematic on Windows. Qt4 can be installed with a binary installer, that will also install an open source (GNU) compiler. Since Riverbank does not provide a binary installer for PyQt4 depending on Python 2.4.* the user has to build it by hand. This can be done using the compiler provided by Qt4. Unfortunately a config file of Qt4 on which PyQT4 is dependant is missing a variable (at the time of writing) that has to be added by the user manually. The detailed instructions can be found in the `INSTALL` file.

2.5.2 Debian Package

To create a debian binary installation package the software packaging program EPM [26] (created by Easy Software Solution) was chosen. It was also chose for its ease of use. It enables a uniform method for building (binary) software packaged for UNIX/Linux systems.

The building of a software package with EPM requires the creation of a *list file*. The following shows the list file for semanticSBML.

```
1  #definition of variables
2  $prefix=/usr
3  $exec_prefix=/usr
4  $bindir=${exec_prefix}/bin
5  $datadir=/usr/share
6  $docdir=${datadir}/doc/semanticSBML
7  $libdir=/usr/lib
8  $pylibdir=${libdir}/python2.4/site-packages
9  $mandir=/usr/share/man
10 $srcdir=/home/foreach/MPG/semanticSBML/trunk/sbmlmerge

12 #main package information
13 %product semanticSBML
14 %copyright 2007 Computational Systems Biology Group (Max Planck Institute for Molecular ...
    ...Genetics)
15 %vendor Computational Systems Biology Group
16 %description Create,Check,Annotate and Merge SBML Documents
17 %description this package includes libsbml 2.3.4 (using xerces) with python bindings
18 %version 0.9.3
19 %readme ${srcdir}/README
20 %license ${srcdir}/COPYING

22 %format deb
23 %requires python2.4
24 %requires python2.4-qt4
25 %requires python-soappy
26 %requires graphviz
27 %requires libxerces27

29 ########libsbml
30 l 755 root sys ${libdir}/libsbml.so libsbml.2.3.4.so
31 f 644 root sys ${libdir}/libsbml.a /usr/local/lib/libsbml.a
32 f 644 root sys ${libdir}/libsbml.2.3.4.so /usr/local/lib/libsbml.2.3.4.so
33 f 644 root sys ${pylibdir}/libsbml/libsbml.py /usr/local/lib/python2.4/site-packages/...
    ...libsbml/libsbml.py
34 f 644 root sys ${pylibdir}/libsbml/libsbml.pyc /usr/local/lib/python2.4/site-packages/...
    ...libsbml/libsbml.pyc
35 f 644 root sys ${pylibdir}/libsbml.pth /usr/local/lib/python2.4/site-packages/libsbml.pth
36 f 644 root sys ${pylibdir}/_libsbml.so /usr/local/lib/python2.4/site-packages/_libsbml.so

38 ########semanticSBML
39 f 755 root sys ${bindir}/semanticSBML ${srcdir}/semanticsbml_gui.py
40 f 755 root sys ${bindir}/semanticSBML-console ${srcdir}/semanticsbml_console.py
41 f 755 root sys ${bindir}/semanticSBML-id2sbml ${srcdir}/semanticsbml_id2sbml.py
```

```
42  f 755 root sys ${bindir}/semanticSBML-check ${srcdir}/semanticsbml_check.py
43  f 755 root sys ${bindir}/semanticSBML-2dot ${srcdir}/semanticsbml_2dot.py
44  f 755 root sys ${bindir}/semanticSBML-exportDB ${srcdir}/semanticsbml_exportdatabases.py
45  f 755 root sys ${bindir}/semanticSBML-reduce ${srcdir}/semanticsbml_reduce.py
46  f 755 root sys ${bindir}/semanticSBML-stabilize ${srcdir}/semanticsbml_stabilize.py

48  #lib
49  f 644 root sys ${pylibdir}/semanticSBML ${srcdir}/semanticSBML/*.py
50  f 644 root sys ${pylibdir}/semanticSBML ${srcdir}/semanticSBML/*.pyc
```

The script starts with defining variables that will be used later on (lines 1-10). Required package meta information is listed in the lines 12-20. Setting the correct version number allows an update of the program with the removal of old program code. In lines 22-25 the package dependencies are set. These can then be automatically resolved by a packet management system. Since semanticSBML is dependant on libSBML a binary distribution of libSBML is included in the package (lines 27-34). The executable scripts of semanticSBML (lines 37-44) are renamed and copied to the `/bin` directory during the installation. All other scripts are place into the global library directory of Python.

Discussion Distributing a binary version of libSBML was welcomed by its developers and a creation of a separate libSBML package was requested. A binary distribution package for libSBML currently does not exist for Linux platforms.

EPM supports most of the mayor Linux distribution and in future development it is aspired to also build packages for other Linux distributions.

2.5.3 Cross Platform Ability

The installation and the functional efficiency of semanticSBML was tested on the platforms: Ubuntu Linux (see Figure 12), OS X (see Figure 13) and Windows (see Figure 14). While working flawlessly on Linux and OS X the graph visualization with Graphviz did not work on the Windows platform.

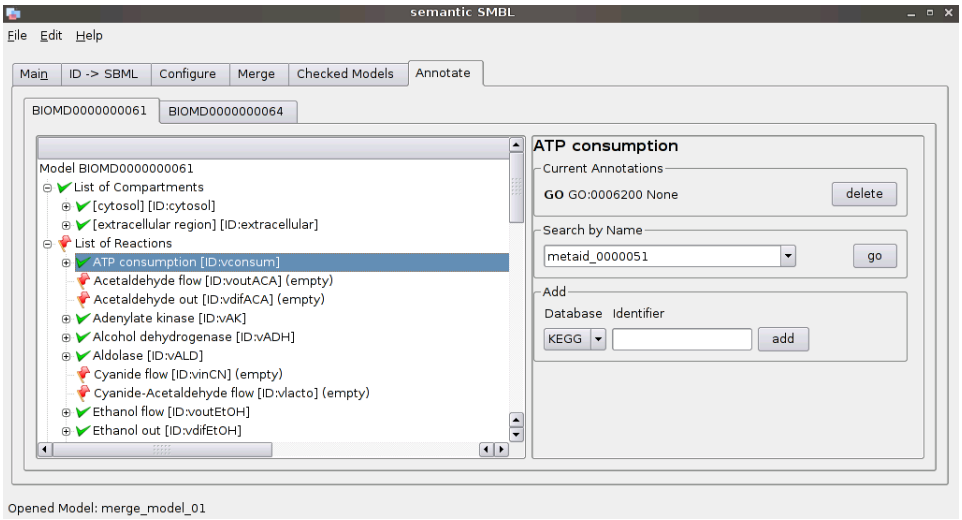

Figure 12: semanticSBML running on Ubuntu Linux with the annotation view open.

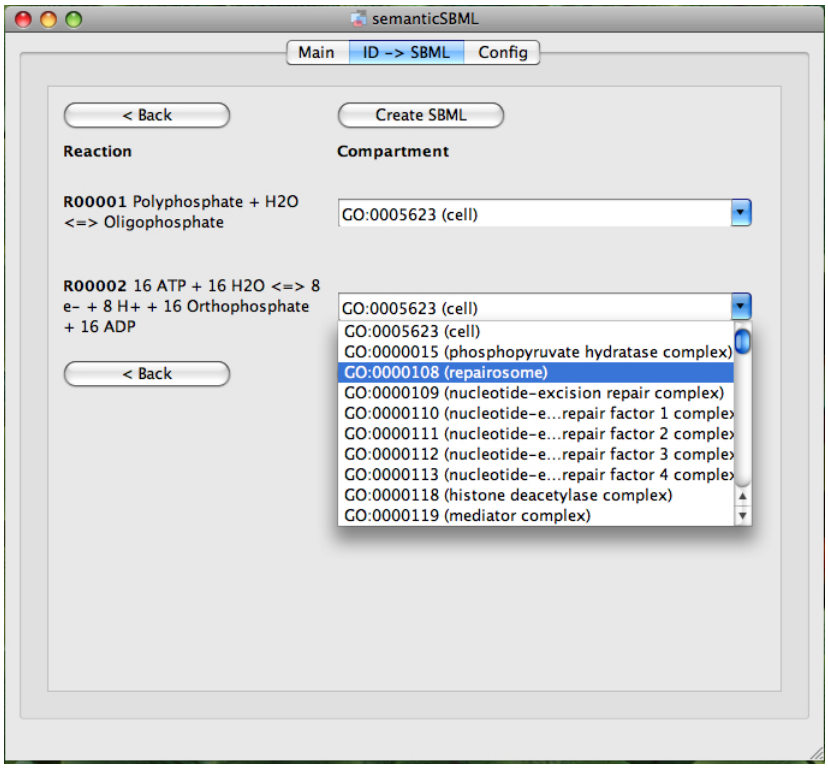

Figure 13: semanticSBML running on OS X (*Apple Inc.*) with the model creation view open.

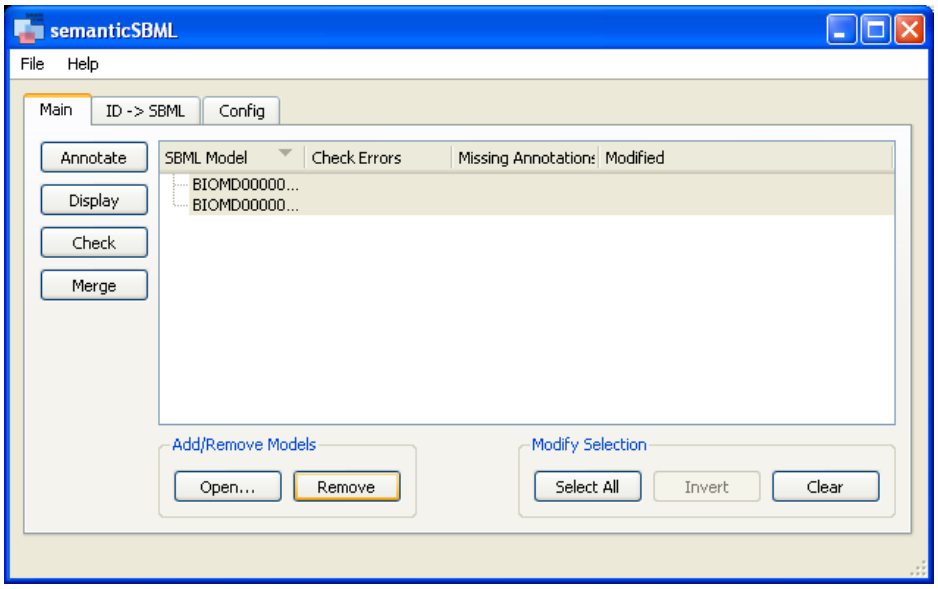

Figure 14: semanticSBML running on Windows (Microsoft Inc.) with the main view open.

3 Phase II

The goal of the second phase was to update the core algorithms for modification of MIRIAM annotations and the merging of models. Since a new version of libSBML was released during the development of the first phase, the new version of libSBML was incorporated and the GUI was updated to fit the needs of the newly developed algorithms.

3.1 Porting to libSBML 3.*.*

The porting to the new libSBML was simple since it was the goal of this phase to rewrite the main algorithms. Some changes had to be applied to the file management for writing loaded models to the hard drive. Functions developed by my colleagues were simplified (while leaving the source code intact) so that they could be adapted over time and did not disturb the main functions.
The parts of the check algorithm that are concerned with the annotation of SBML elements were removed. In the future these parts can be replaced by errors that are raised by the new annotation classes.

The graph visualization algorithm depends on non backwards compatible functions of the libSBML and had to be disabled.

3.2 Annotate

One of the main tasks of semanticSBML is the annotation of SBML models with MIRIAM annotations. The MIRIAM annotation does not only play a role in the merging of models but could also become a standard for publicly released SBML models.

3.2.1 The MIRIAM annotation

Introduction The SBML format allows the annotation of elements and of the whole model. An annotation can be e.g., two dimensional coordinates of icons that represent a reaction in a graphical visualization by the popular tool CellDesigner [27]. Annotations are optional and their format can be specified by their creator as long as they follow the external XML standard RDF [28]. As mentioned in the introduction Section 1, MIRIAM was created as an effort to ensure the quality of a model and enable a fast entity recognition. MIRIAM itself is a proposed framework of rules that consits of two parts. The first part describes the syntax and semantics a model description should follow. The second part is an annotation scheme. This annotation scheme can be applied to SBML elements when encapsulating it into a RDF element. The RDF format is used to create semantic statements about an object using a subject-predicate-object expression. The subject is in this case a libSBML element (a biological entity). The object is an external resource that holds a reference description of the entity. The external resource is given by a pair that consits of an URI [29] that is joint with the symbol "#" and an identifier string to from a URL [30]. The URI representing a data resource that provides a description of a biological entity which can be found with the identifier string. The predicate that describes the relationship between the subject and the object is given by

BioModels qualifier elements [31]. The following example shows a MIRIAM annotation in SBML (the example is part of a SBML document "..." denotes the rest of the document).

```
1  ...
2  <compartment metaid="metaid_0000075" id="cytosol" size="1">
3      <annotation>
4          <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:bqbiol="...
            ...http://biomodels.net/biology-qualifiers/" xmlns:bqmodel="http://biomodels.net/...
            ...model-qualifiers/">
5              <rdf:Description rdf:about="#metaid_0000075">
6                  <bqbiol:is>
7                      <rdf:Bag>
8                          <rdf:li rdf:resource="http://www.geneontology.org/#GO:0005829"></...
                            ...rdf:li>
9                      </rdf:Bag>
10                 </bqbiol:is>
11             </rdf:Description>
12         </rdf:RDF>
13     </annotation>
14 </compartment>
15 ...
```

The example above shows the SBML element `compartment`. The element is annotated with a reference to the Gene Ontology identifier for cytosol. The MIRIAM annotation in human words states "The element *compartment is* the identifier *GO:0005829* (cytosol) of the database *http://www.geneontology.org/* (Gene Ontology)". The table below explains the important sections of the example and introduces the terms that will be used in the following description.

Line	**Term**	**Example**
2	Entity	`compartment`
6	Qualifier	`bqbiol:is`
8	Database/Data-Type	http://www.geneontology.org/
8	Identifier	GO:0005829

For a complete description of the example the SBML and RDF specifications should be used.

The originally proposed term data-type will be referred to as *database*. The word *annotation* will be used synonymous for MIRIAM annotation.

3.2.2 Concept

The main concept of the annotation algorithm is that is a simplified abstraction of a SBML model (see Figure 15). The main idea is that a *model* consists of *elements* that can be *annotated*. The type of an element is defined by an attribute and not by the element itself like in SBML. In the new annotation concept a model consits of a collection of elements and elements consit of a collection of annotations. Thus the model depends on its elements and elements depend on its annotations. However an annotation can be used independently from and element and an element can be used independently from a model. The construction of the model and element depend on the SBML whereas the annotation can be created independently from SBML. The annotation was constructed this way on purpose to enable a possible reuse outside of semanticSBML.

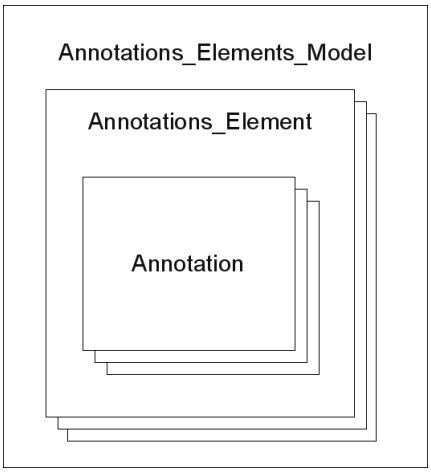

Figure 15: The concept of the new annotation algorithm. Each level of the abstraction used for the annotation algorithm can be used independently. The `Model` and the `Element` level depend on SBML whereas the `Annotation` level is SBML independent.

3.2.3 Features

The basic idea of the RDF format was introduced in Section 3.2.1. The main features of the algorithm are introduced by listing the available values that the *subject*, *predicate* and *object* of the RDF annotation object can have.
Subject The new annotation algorithm supports annotations to the following libSBML elements (see Appendix B): *species*, *compartment*, *reaction*, *parameter*, *assignment rule*, *rate rule*, *algebraic rule*, *event* and the model itself. The support of libSBML elements is dependant on the `listofresources.xml` file and can be extended by the user. This will be described in the following paragraphs.
Predicate It is able to modify all current BioModels qualifiers.
Model-Qualifiers:

is
: The modelling object represented by the model component is the subject of the referenced resource. For instance, this qualifier might be used to link the encoded model to a database of models.

isDescribedBy
: The modelling object represented by the component of the encoded model is described by the referenced resource. This relation might be used to link a model or a kinetic law to the literature that describes this model or this kinetic law.

unknown
The qualifier is unknown. This is not a part of the BioModels qualifiers but of the libSBML. It is needed since a qualifier is mandatory for a annotation.

`Biological-Qualifiers`:

is
The biological entity represented by the model component is the subject of the referenced resource. This relation might be used to link a reaction to its exact counterpart in KEGG or Reactome for instance.

hasPart
The biological entity represented by the model component includes the subject of the referenced resource, either physically or logically. This relation might be used to link a complex to the description of its components.

isPartOf
The biological entity represented by the model component is a physical or logical part of the subject of the referenced resource. This relation might be used to link a component to the description of the complex it belongs to.

isVersionOf
The biological entity represented by the model component is a version or an instance of the subject of the referenced resource.

hasVersion
The subject of the referenced resource is a version or an instance of the biological entity represented by the model component.

isHomologTo
The biological entity represented by the model component is homolog, to the subject of the referenced resource, i.e. they share a common ancestor.

isDescribedBy
The biological entity represented by the model component is described by the referenced resource. This relation should be used for instance to link a *species* or a *parameter* to the literature that describes the concentration of the *species* or the value of the *parameter*.

unknown
The qualifier is unknown. See "unknown" model qualifier.

Object The database support is not as easy to list as the support for the libSBML elements and the qualifiers. It depends on two components of semanticSBML. The first component is the internal database and the second one is the XML file `listofresources.xml` which is included in the program package.

The internal database provides human representations of identifiers as well as information about the identity of identifiers from different external databases. It is used in the search for annotations. The internal database supports the

databases: Gene Ontology, KEGG, Reactome, ChEBI [32], CAS [33] and 3dmet [34]. The internal database was created with SBMLmerge and will thus not be discussed in detail - only the functions that are used by the merge algorithm may be mentioned.

The `listofresources.xml` file provides a list of database-URIs, -names -idpatterns etc. It is used to show human readable representations of databases, create hyperlinks to the databases and do a basic check of the correctness of identifiers. The first version of the `listofresources.xml` file was provided by the BioModels team and was incorporated into semanticSBML since it provides a flexible lightweight method to incorporate new databases for the annotation. The file consits of a list of `resource` elements. Each `resource` represents one database. The structure will be explained using the following example.

```
1  <resource
2          name="EC code"
3          uri="http://www.ebi.ac.uk/IntEnz/"
4          alternateUris='http://www.ec-code.org/'
5          location="http://www.ebi.ac.uk/IntEnz/"
6          action="http://www.ebi.ac.uk/intenz/query?cmd=SearchEC&amp;ec="
7          elements="assignmentRule rateRule algebraicRule reaction event"
8          idPattern="^(\d+|\d+\.(-|\d+)|\d+\.\d+\.(-|\d+)|\d+\.\d+\.\d+\.(-|\d+))$"
9          />
```

Line	Description
1	Opening of the `resource` tag
2	Human readable name of the database
3	Primary URI (part of the MIRIAM annotation)
4	Alternative URI that can be used to reference the same database
5	World wide web location of the database
6	URL that can be combined with an identifier to create a hyperlink referring to a description of the annotation.
7	Space separated list of libSBML elements that can be annotated using this database.
8	Regular expression pattern that all identifiers of this database must follow. The pattern is used for a basic check of the annotation identifiers.
9	Closing of the `resource` tag

The `listofresources.xml` file is located in the semanticSBML subdirect which is located in the home directory of the user that installed semanticSBML and thus in a easily accessible location. The users of semanticSBML are encouraged to edit the file to fit their needs.

3.2.4 Implementation - API

The implementation of the annotation algorithm follows closely the concept previously described in Section 3.2.2. The objects described in the concept match exactly the classes of the annotation algorithm. The class diagram can be seen in Figure 16. The interface functions and some internal functions will be shown in the following listing.

The `Annotation` class represents a single MIRIAM annotation. It consits of methods to get and set the variables: database, identifier, qualifier, qualifier type. When setting these variables multiple checks are executed to verify their correctness. The class uses the external file `listofresources.xml`. In addition the class provides functions for comparison of annotations and different representations of the annotation.

Function
: Description

`__init__`
: Set the resource (database and identifier) and qualifier (qualifier and qualifier type).

`__eq__`
: Equality operator. If database and identifier are the same return `True`.

`__str__`
: Return the annotation resource as specified by the proposed MIRIAM annotation standard: URI#ID.

`getName`
: Return a human readable string representation of the annotation - if the resource can be found in the internal database or an empty string if the resource can not be found.

`getURIAction`
: If possible return a hyperlink to find the referenced element on the world wide web.

`setQualifier`
: Set the qualifier of the annotation - libSBML encoded the biological-qualifiers in numbers between 0 and 7 and model-qualifiers between 0 and 2 and qualifier-types between 0 and 2. Both numbers and string representations (e.g., hasPart) are allowed as input for the qualifier and the qualifier type. If the qualifier is not recognized an error is raised.

`setLink`
: Set the database and the identifier of the `Annotation` class. As input for the database a URI (e.g., http://www.geneontology.org) or name (e.g., Gene Ontology) are both accepted. The input allows setting a flag upon which the insertion of unknown databases (known ones are specified in `listofresources.xml`) raise an error. If the identifier pattern is known for the inserted database and the inserted identifier does not match this pattern an error is raised (see `_checkIdPattern`).

`_checkIdPattern`
: This function checks if a regular expression pattern for identifiers of a given database can be found (`listofresources.xml`). If a pattern is found the inserted identifier is matched against the pattern. If the pattern matches the function returns `True`, if it does not match the function returns `False`.

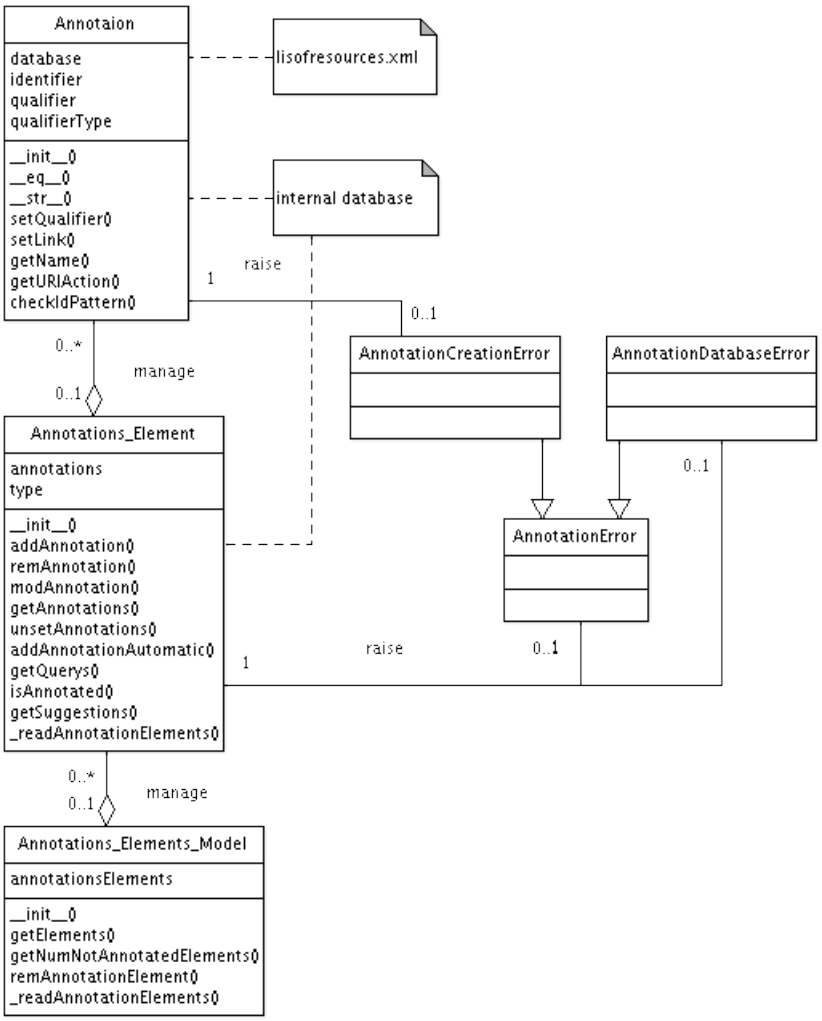

Figure 16: Class diagram of the improved annotation algorithm. The `Annotation` represents a single annotation. The `Annotations_Element` represents a SBML element. The `Annotations_Elements_Model` represents a SBML model.

The `Annotations_Element` class represents one annotatable object in a model (a biological entity / libSBML base element). It is a container for `Annotation` class instances. It has functions to add, remove and modify annotations.

Function
: Description

`__init__`
: Read MIRIAM annotations, type, id, metaid and name from libSBML element. Only MIRIAM annotatable libSBML elements are allowed as input.

`_readAnnotations`
: Internally used function to read all MIRIAM annotations (from `CVTerms`) of the inserted libSBML element

`isAnnotated`
: Return if the element contains MIRIAM annotations.

`addAnnotation`
: Add a MIRIAM annotation to the SBML element represented by this class instance. In libSBML versions <3.0.2 the adding of identical annotations will create (two) separate identifiers. As a result of this work libSBML prevents this in later version. (The function also checked if a `CVTerm` with the same qualifier already exists and add annotation to this `CVTerm` - this functionality was discontinued since libSBML already provides it.)

`modAnnotation`
: Modify the qualifier of an annotation.

`remAnnotation`
: Delete an annotation from libSBML element and resynchronized the internal list of annotations with libSBML element.

`unsetAnnotations`
: Delete all annotations of this element. This function is not present in the current libSBML but might be added in future versions. It was created due to the behavior of libSBML <3.0.2 described in `addAnnotation`.

`getAnnotations`
: Return a list of `Annotation` class instances. These instances should be used to add or remove annotations.

`getQuerys`
: Return a list of the `name`, `id` and `metaid` value of the libSBML element. These can be used to query annotations from the internal database.

`getSuggestions`
: Get a list of `Annotation`s by querying the internal database. If no query is specified as input the function `getQuerys` is used. A switch disables the fuzzy database search, then only exact matches are returned.

`addAnnotationAutomatic`
: Check if the element is already annotated (using `isAnnotated`). If is not annotated, check if an annotation can be found using `getSuggestions` (exact matches only). Add annotation(s) that were found to this element (using `addAnnotation`).

The `Annotations_Elements_Model` represents a complete model. It is a container for `Annotations_Element` instances. Its functions are limited since it is mainly used as a data container.

Function
: Description

`__init__`
: Read all elements from a SBML model that can be annotated and create a list of `Annotations_Elements` (using `_readAnnotationElements`).

`getNumNotAnnotatedElements`
: Return the number of elements that have no MIRIAM annotations.

`getAnnotationElements`
: Return the list of `Annotations_Element` (that can contain MIRIAM annotations) available in this model.

`remAnnotationElement`
: Remove an element.

`_readAnnotationElements`
: Go through all MIRIAM annotatable elements in a libSBML model and create `Annotations_Elements`.

In addition to these classes the `Merger` class of SBMLmerge was replicated for backwards compatibility. It is a simple wrapper class. Most of its functions can be found (with different names) in one of the classes above.

3.2.5 Implementation - Integration

The following example shows the usage of the model class. It is initialized with a libSBML `Model` instance. `AnnotaionErrors` have to be caught in case the Model has elements with faulty annotations.

```
1  import libsbml
2  from semanticSBML.annotate import *

4  document=libsbml.readSBMLFromString(open('mymodel.xml','r').read())
5  try:
6      aem=Annotations_Elements_Model(document.getModel())
7  except AnnotationError,e:
8      print e
9  else:
10     print aem.getNumNotAnnotatedElements()
```

The `Annotations_Element` class is initialized with a libSBML element in this case a *species* element. Like in the initialization of the model the element might raise an exception if the annotations of the element are faulty. On a successfull initialization the number of non annotated models is printed to the screen.

```
11 try:
12     ae=Annotations_Element(list(document.getModel().getListOfSpecies())[0])
13 except AnnotationError,e:
14     print e
15 else:
16     print ae.isAnnotated()
```

`Annotation` instance can be created in different ways. In line 17 and 18 identical annotations are created using different input. In line 17 human readable representations are used in line 18 the URI and the libSBML numbers of the BioModels qualifiers are used (see Section 3.2.4). Line 19 creates an annotation for a model.

```
17 a1=Annotation('Gene Ontology','GO:1234567','bio','is')
18 a2=Annotation('http://www.geneontology.org/','GO:1234567','1','0')
19 a3=Annotation('BioModels','BIOMD0000000001','model','is')
```

The annotation objects created in the example above can be used to add annotations to a model.

```
20 ae.addAnnotation(a1)
```

The adding of the annotation will modify the libSBML model instance. To save the changes persistently the model has to be written to a file on the harddisk by using libSBML functions.

3.2.6 Annotation GUI

In comparison to the first version of the annotation GUI visible new features are the qualifier modification widgets and annotation resources as hyperlinks. Figure 17 shows a screenshot of the new annotation GUI with numbers indicators for the different features that will be explained in the following legend.

Legend of Figure 17:

1. Choice widget with a list of biological and model qualifiers.
2. Hyperlink to a world wide web location that will open a external browser.
3. Clicking the "change" push-button will set the qualifier of the annotation (`modAnnotation`) that was chosen (see 1).

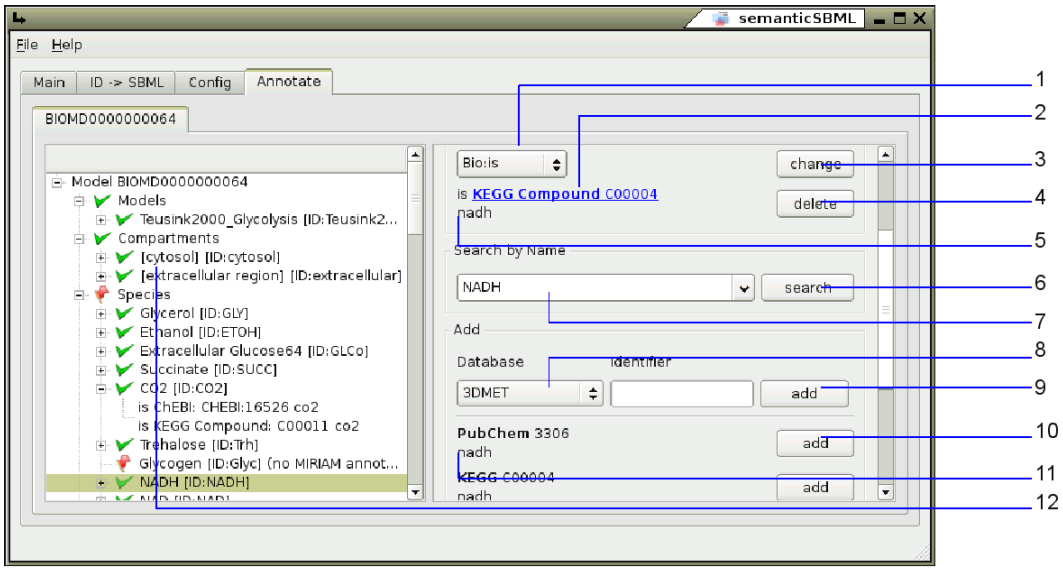

Figure 17: GUI to the new annotation algorithm. The legend can be found in Section 3.2.6

4. Remove this annotation from the model element (`remAnnotation`).

5. Human readable representation of the annotation (`getName`).

6. Start a fuzzy search on the internal database for the query inserted (see 7) (`getSuggetsions`). The results will appear in 11.

7. Editable choice widget with suggestion of queries for a database search (`getQuerys`).

8. Drop-down-box widget with a list of known databases (`listofresources.xml`) that can be used to annotate the current element.

9. Clicking this push-button an attempted will be made to add the manually constructed annotation to the current element (`addAnnotation`). If the inserted identifier does not match the regular expression pattern of identifiers of the database (`_checkIdPattern`) a pop-up appears displaying an error message.

10. Add the annotation that is displayed next to the push-button to the current annotations of the element (`addAnnotation`). The new annotation will have the biological qualifier `unknown`.

11. Annotation that was found in the internal database. The annotations that are dispayed in this area can be found using two different methods. The first method is an initial search that is conducted when the element is opened (`getSuggestions`). The second method is a manually search (see 6).

12. The treeview displays all annotatable model elements. The green icon indicates that the element is MIRIAM annotated; the red icon indicate that element is not MIRIAM annotated. The tree item displays the value of the `name` and `id` attribute of the `Annotation` element / libSBML element. If the `name` is not set but a MIRIAM annotation is available (like in the example) the internal database is used to diplay a human readable string representation of the first annotation of the element in square brackets (`getName`). Expanded nodes display all annotations of an element.

For adding annotations automatically the nodes of the treeview (left hand side of the annotation widget, see 12) on the first level e.g., "Species" has to be selected. This will create a menu on the right hand side of the widget which contains a push putton that starts the automatic annotation of all elements that are child nodes of the currently selected tree node. (e.g., all `Species` elements).

3.2.7 Discussion

The new annotation algorithm delivers a new API for the manipulation of MIRIAM annotations. Its design is flexible and includes all current BioModels qualifiers. The algorithm allows the integration of databases by the user. It can be argued that this will endorse the use of none accepted databases, however semanticSBML addresses a professional audience and will therefore only guide but not restrict a user.

The design of the internal database has several disadvantages e.g. a complicated access to the comparison of identifiers and missing relations between identifiers of one database. An improvement of the internal database would also improve the annotation process.

The semanticSBML annotation interface should help the systems biology community to accept MIRIAM annotations. One of many new vistas that is opened up by an API for the manipulation of MIRIAM annotations is shown in an experiment in Section 4.1.

3.3 Merge

As already mentioned in the introduction (Section 1) semanticSBML uses its own abstraction of a systems biology model. The abstraction is based on the idea (see Figure 3) that a model consists of biological entities. A biological entity (`BioEntity`) can be identified. It is described by a biological quantity (`BioQuantiy`) and the model makes statements about the entity (`ModelStatement`). This abstraction differs from the abstraction of the SBML format.

It was created for the following reasons: The experiences with SBMLmerge showed that object dependencies in the merging process were not clear. The merging of elements with different types should be allowed. The conflict resolution of SBMLmerge did not differentiate the severity of a conflict. In addition to that the abstraction allows a simple reuse of functions that are needed during the merge process.

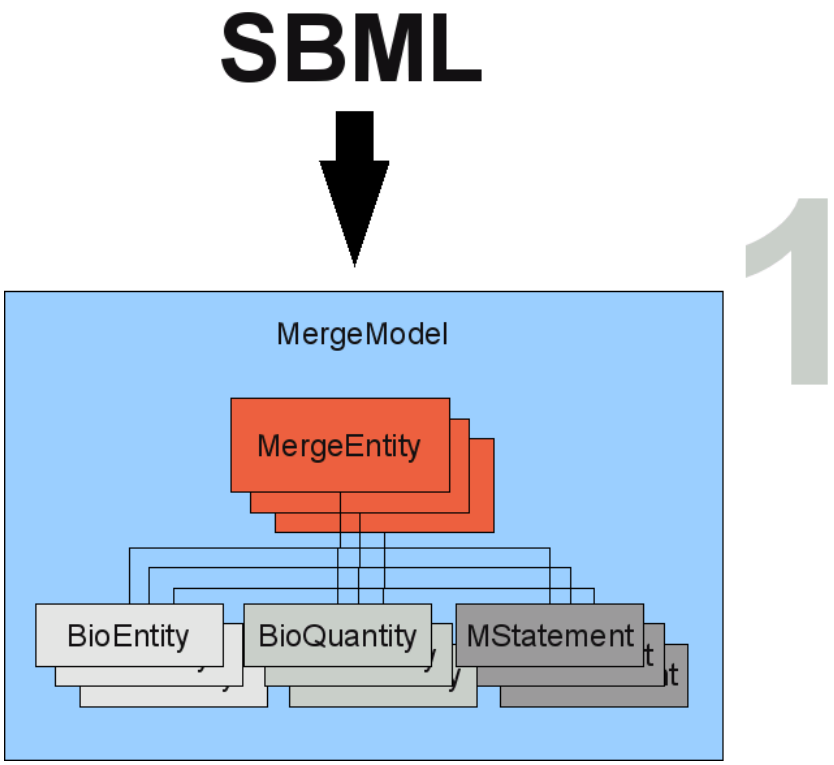

Figure 18: Merge Concept: In the first step (large grey number) of the semanticSBML merging algorithm, the SBML document is translated into semanticSBMLs own abstraction.

3.3.1 Concept

Next to the model abstraction the the mergin process in semanticSBML uses further concepts that will be introduced in this section. In the following class names (written in `typewriter font`) are used along with artificial names for datastructures. LibSBML elements are written in *slanted font*.

The first step is the translation of a SBML model to a semanticSBML model (see Figure 18 step 1; the steps are marked with large grey numbers). On this account the libSBML base elements (see Appendix B) *compartment species parameter reaction* (these elements are considered as mergable biological entities) are translated into `MergeEntitys`. The `MergeEntity` class can be viewed as meta SBML element since it contains all of the attributes the different types of SBML elements can contain. In some cases an attribute can also be a SBML base element. The `MergeModel` stores all `MergeEntitys` of one libSBML model.

The translation is repeated for all models that should be merged (see Figure 19 step 2). All elements in each model are compared pairwise with the elements (of matching type) in the other models (see Figure 19 step 3). The algorithm checks for biological identity with the help of MIRIAM annotations as well as

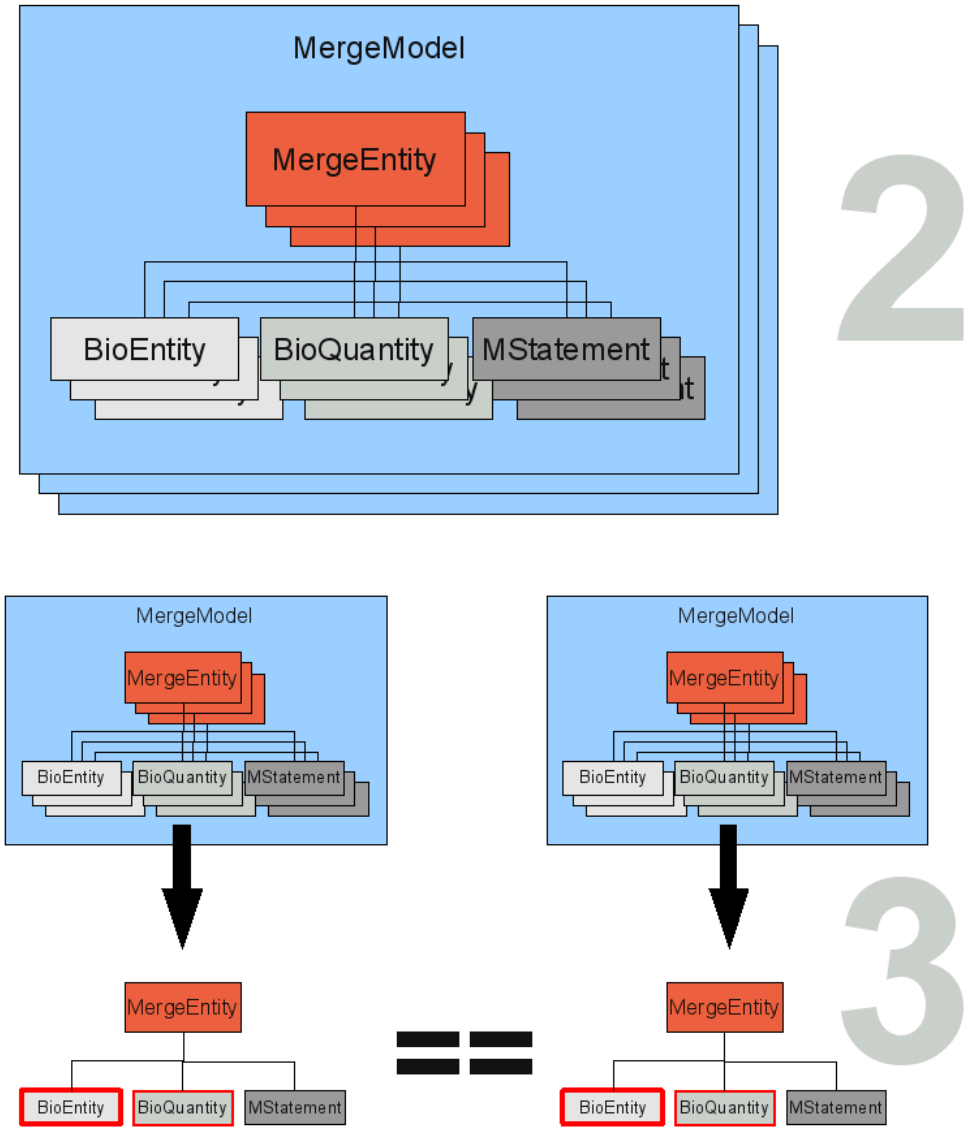

Figure 19: Merge Concept: In the second step the translation (Figure 18 step 1) is repeated for all documents that should be merged. In the third step all mergable elements of each model are compared in a pairwise manner.

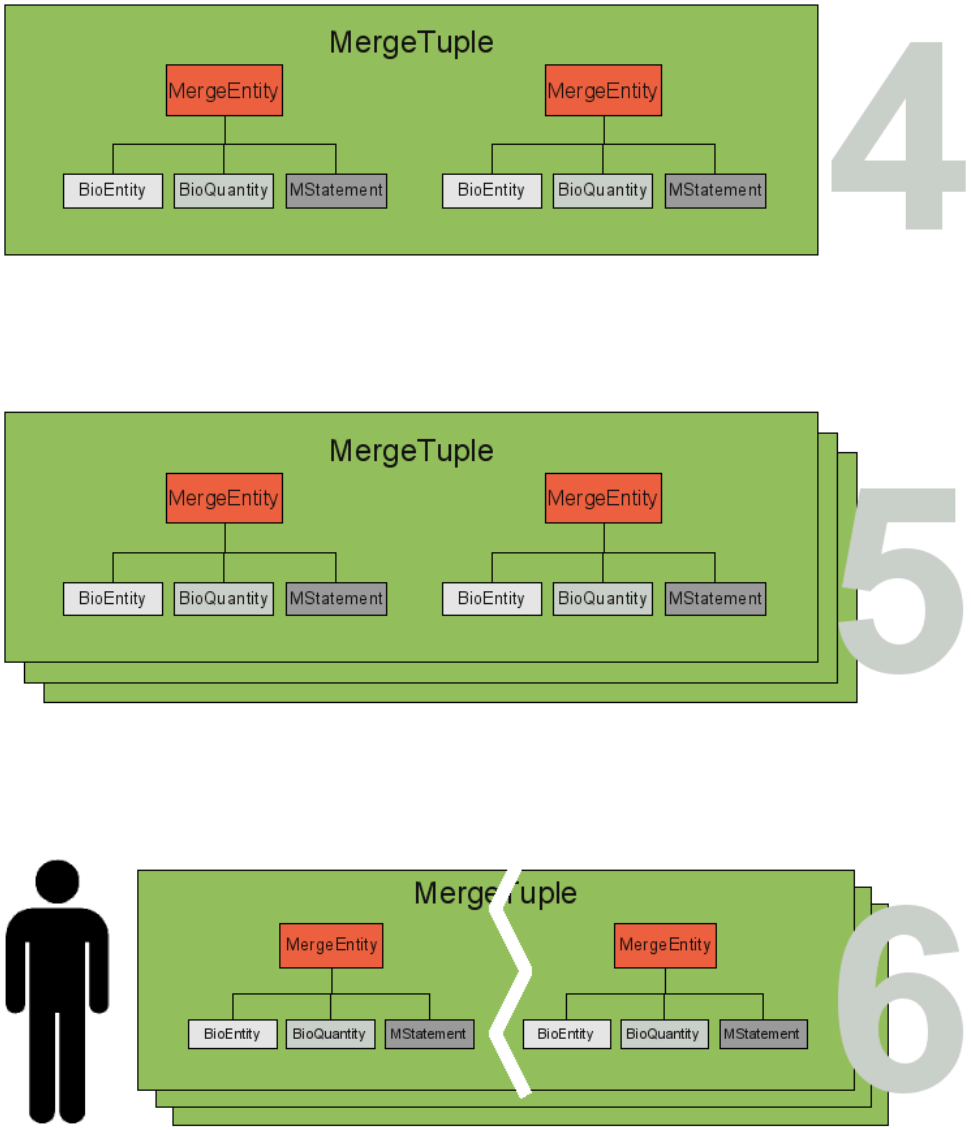

Figure 20: Merge Concept: If in the third step (Figure 19) duplicate entities are found a `MergeTuple` is created. During the comparison process a list of tuples is build up (step 5) that can be modified by a user (step 6).

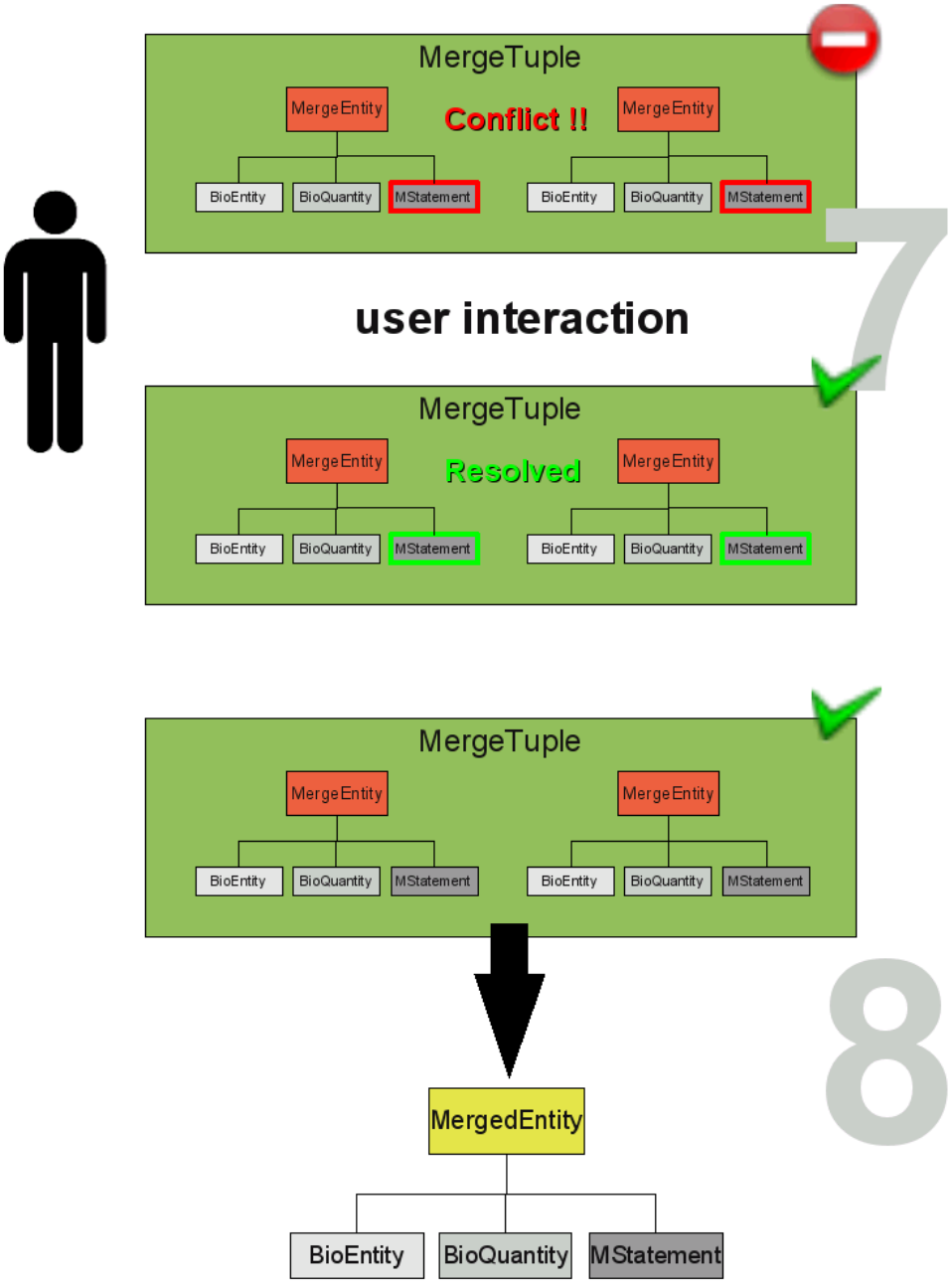

Figure 21: Merge Concept: Tuples of biological entities can contain conflicting values. The conflicts must be resolved by user interaction (step 7). From a tuple a new entity is created that is the merged entity of all entities in the tuple (step 8).

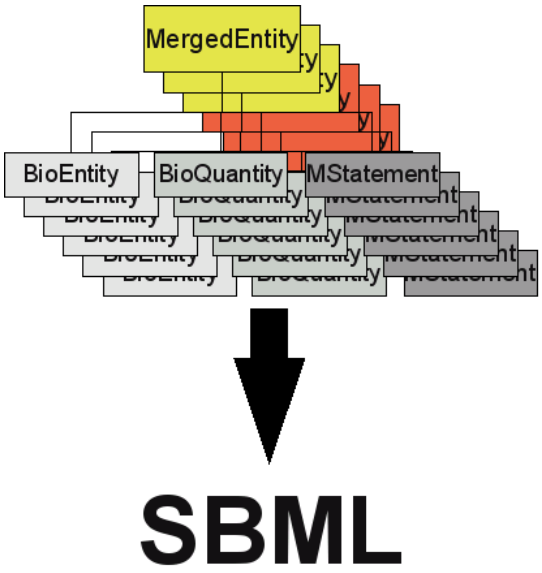

Figure 22: Merge Concept: The collected `MergeEntity`s and `MergedEntity`s integrate all information of the models that were merged. The merging process ends with a retranslation of the semanticSBML model into a SBML model.

structural identity (location of physical biological entities e.g., ATP in cytosol - ATP in mitochondrion).

If duplicate entities are found they are stored in a `MergeTuple` (see Figure 20 step 4). The `MergeTuple` is a smart container that only allows the storage of one entity from each model (an aggregation of entities is not allowed). It also decides which element belongs in the container. A list of `MergeTuple`s is generated (see Figure 20 step 5). The user can manually modify this list since a correct matching of the duplicate entities can not be guaranteed (see Figure 20 step 6). Even though it is not desired, this also enables the manual merging of non annotated models.

The entities in the `MergeTuple` may contain conflicting values. Most conflicting values can be resolved by choosing a value from a list e.g., choosing the correct initial amount of the ATP (see Figure 21 step 7). However conflicts in the `BioEntiy` (determining the identity of the entity) or the `BioQuantiy` (describing the entity) show that there is a severe disagreement between the entities. The resolution of severe conflicts may need more than a simple choice and should often lead to the destruction of the tuple. The user can however solve every conflict and thus mark the `MergeTuple` as *resolved*. From a resolved or non conflicting `MergeTuple` a combined element is created called the `MergedEntity` (see Figure 21 step 8). The `MergedEntity` has the same properties as a `MergeEntity` and has only implementational importance.

From the list of all `MergedEntity`s and `MergeEntity`s a merged SBML model

can be created (see Figure 20 step 8). The merged SBML model integrates all information of the inserted models.

The semanticSBML merge algorithm tries to realize this concept of the merge process however some changes had to be made for an enhanced usability. Since this description in this section is a simplification Section 3.3.2 will describe the actual implementation.

3.3.2 Implementation

The merging algorithm is interface based. A class diagram of the merge algorithm can be seen in Figure 23. The interface to the merge algorithm is provided by the `Merger` class. The `Merger` class is initialized with a list of libSBML document instances.

Translation The first step in the merging of the model is the translation of the SBML model into the semanticSBML datastructure. Figure 24 shows the basic mapping of SBML base elements to the semanticSBML datastructure. The SBML base elements *compartment*, *species*, *parameter* and *reaction* are viewed as mergable biological entities and are therefore translated into `MergeEntity`s. Their values and subelements are stored in either the `BioEntiy`, the `BioQuantity` or `ModelStatement` class. The three classes are stored in the `MergeEntity` class along with other general properties.

Identification The `BioEntiy` class is used to identify the biological object. It stores an elements MIRIAM annotation using a `Annotations_Element` (see Section 3.2).

Description The `BioQuantiy` class describes an element. One if its attributes is *type* of the BioQuantity. The type in the case of a physical biological entity (*species*) determines if it is an *amount* or a *concentration* (SBML property `hasOnlySubstanceUnit`), in the case of a *compartment* it is the dimension of the *compartment* (e.g., the nucleus is a three dimensional *compartment* whereas the cellwall is a two dimensional *compartment*). The class also stores *reaction participants* (*reactand*, *product* and *modifier*) for *reaction* elements. In addition to that the class stores the *unit* and *location* of an element. The location attribute contains a pointer to another `MergeEntity` (of the type *compartment*). The unit is stored as a custom dictionary type. The storage of units is redundant in comparison to SBML (*unit*s are a base elements and can be predefined) but easier to use since standard units are resolved and can be used along with custom units.

Statements The `ModelStatement` class holds the largest amount of information. A part of the attributes can be summarized as *simple attributes* e.g., the initial amount (`float`) of a *species* or the flag for reversible *reaction*s (`bool`). The simple attributes are either of type `float` or `bool`. The class also stores *mathematical statements*. In the case of a *reaction* the mathematical statement is the *kinetic law* and in the case of *compartment*, *species* and *parameter* it can be a *rule* or *initial assignment*. The mathematical statements are stored as copies of libSBML element instances. The mathematical statements

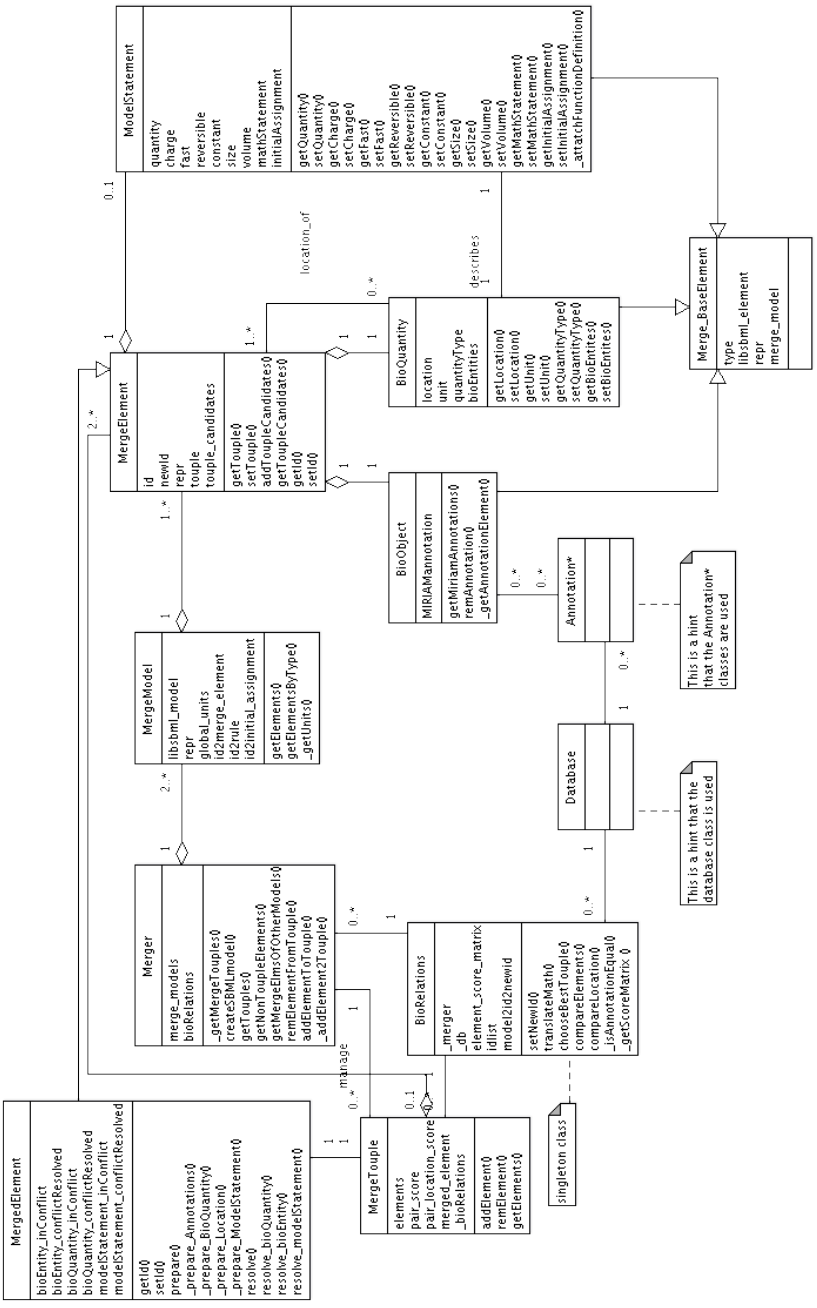

Figure 23: Class diagram of the improved merging algorithm. The `Merger` class is the interface to external classes. A model is represented by the `MergeModel` class. It contains a row of `MergeElements`. Each `MergeElement` consists of one `BioEntiy`, one `BioQuantiy` and one `ModelStatement`. The `MergeTuple` is a smart container for `MergeElements`. The `BioRelations` class provides functions that are used in the context of the merged model and that use external resources.

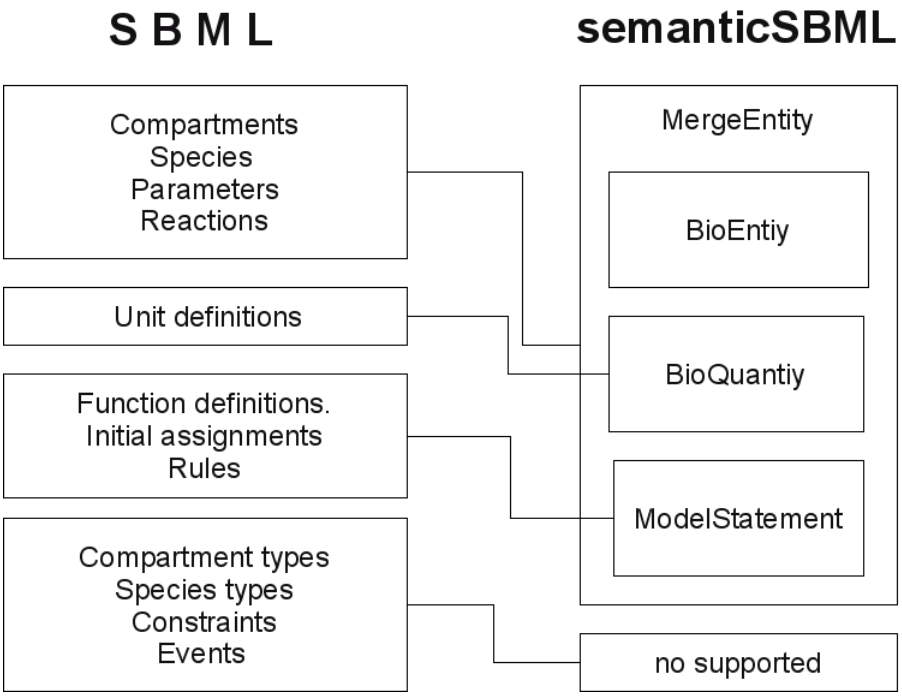

Figure 24: The figure shows the mapping of the SBML base elements (left) to the semanticSBML datastructure (right).

can reference other mathematical statements (the SBML *function definition*s base element). Copies of the referenced *function definition*s are attached to the mathematical statements. Similar to the storage of units this can lead to a redundancy .

The `BioEntity`, `BioQuantity` and the `ModelStatement` classes are all derived from a base element that stores information about the type and the model they are derived from. This is needed for the correct initialization of all attributes.

Compare For a pairwise comparison of all `MergeEntity`s the entities of each `MergeModel` are traversed. The compassion itself is executed by a function of the class `BioRelations`. The `BioRelations` class holds a triangular *score matrix* for all biological BioModels qualifiers matched against itself ("is" vs "is", "is" vs. "has part" ...). The MIRIAM annotations of each element are compared with the MIRIAM annotations of the other element. If two annotations are found to be equal by identity (`Annotation __eq__` operator see Section 3.2.4) or by belonging to the same group in the internal database the score of the qualifiers is looked up in the score matrix and returned (e.g.,"is" vs "is" returns 10). All scores are added up and returned as an overall score. If the overall score is higher than 0 an attempt is made to create a tuple containing both elements or adding one of the elements to an already existing tuple. If both elements are in a tuple, one of the elements is added to the best matchin tuple. The attempt of adding an element to a tuple can fail if the tuple contains an element of the

same model with a higher match score. In addition to that the match score of the elements location (*compartment* element) is looked up and compared with the *location score* of a competing element. The *compartment* score is always the divisive score if competing elements exist, since it is important for the structural identity of an elements in the merged model. Matching elements are stored in a `MergeTuple`. All elements that matched but were not merged retain links to eachother. These links are used in following merging process to obtain a list of *similar* elements. The discussion in Section 3.3.4 will address the matching problem further.

Differences to the Concept The concept of the merging process in Section 3.3.1 described that after the compare step (Figures 19 and 20 steps 3-5) the manual manipulation of the `MergeTuple` list would follow (Figure 20 steps 6). Only after these steps the conflict resolution (Figure 21 steps 7) and the generation of `MergedEntity`s (Figure 21 steps 8) follows. However in the implementation there is no separation between the updating of the `MergeTuple` list and the merging of tuples. The implementation continues with the generation of an *randomly* merged model. The random merged model provides the user with valuable information for the manipulation of the `MergeTuple` list. On this account the following paragraph describes the merging process before the updating of the `MergeTuple` list.

Merging Before elements can be merged the identifiers of all elements have to be collected and stored in a global list. The list is stored in the `BioRelations` class. This is followed by the creation of `MergedEntity`s from `MergeTuple`s. The `MergedEntity` class is derived from the `MergeEntity` and contains a copy of all the values of a *random* `MergeEntity` of the `MergeTuple`. The `MergeEntity` class extends the `MergeEntity` class by boolean variables that indicate conflicting values between elements of the `MergeTuple` (*conflict flag*s), dictionaries containing the possible choices for conflicting values as well as functions to *generate* the *conflict dictionaries* and functions to *resolve* conflicts.

Conflict Resolution To generate a conflict resolution dictionary the attribute values of each element are checked for equality. For complicated data types the equality is determined by the equality of their string representations. The conflict resolution dictionaries contain only values that can be used directly in a merged model. This means all referenced element identifiers are updated before they are added to a dictionary. Mathematical statements undergo an *identifier update* with the help of a function of the `BioRelations` class. If the referenced elements are about to be merged the `MergedEntity` is reference instead.

The merging of the `BioEntity`s MIRIAM annotations is an aggregation of all non-identical annotations. The aggregation can lead to contradicting annotations, that is two annotations with the qualifier "is" reference the same database with different identifiers. If this problem occurs a flag set to indicate the conflict. The user must solve the problem by deleting non applicable annotations. If the problem still exist on a resolution attempt of the user, a `MergeError` is raised.

For the resolution of conflicts that can be solved by a choice. The set func-

tions of the `BioEntity`, `BioQuantity` and `ModelStatement` classes are used. The set functions do not only set the values of the semanticSBML object instance but also of the underlying libSBML object instances. In addition to that the semantic correctness of the chosen values is checked. A semantic error occurs if the `BioQuantiy` type is amount and but an initial concentration is set. An invalid semantic will raise a `MergeError`. If the resolution of all conflicts is successful a flag is set that indicates that the problem is solved (*resolution flag*).

Element Types For the merging of elements of different types the following combinations are allowed: *parameter* with *species* and *parameter* with *compartment*. The type of the resulting element always prefers the more complex element type (*species* and *compartment*). This is similar to a strategy that is used in the initial creation of `MergedEntity`s that is if one of the elements merged elements contain more information this information is used in the creation of the `MergedEntity`.

Manual Matching The manual matching contains two basic cases. An element can be removed from a tuple or an element can be added to a tuple. The removing can result in the destruction of the tuple. The adding can also result in the destruction of the original tuple and it can result in the creation of a new tuple. If an element is added to a tuple where an element of the same model already exists the element is removed from the tuple. The manual matching causes a structural change in the merged model which means that all objects depending on any of the involved elements (e.g., that have a mathematical equation pointing to the element) have to be updated. The first implementation to resolve this problem was a global recreation of all `MergedEntity`s. While it solved the problem it proved to be computationally too expensive.

Translation The final step is the translation of the semanticSBML model back into a SBML model. The translation is only executed if all `MergedEntity`s are conflict free or have resolved conflicts. If this is not the case a `MergeError` is raised listing all elements that are still in conflict. The translation starts with a creation of an empty SBML model. It continues with the collection of all SBML base elements that are attributes of `MergedEntity`s (*rule*s, *initial assignment*s and attached *function definition*s). These elements are directly added to the model. After an update of the identifiers of referenced elements of the non merged `MergeEntity`s the SBML base elements of non merged `MergeEntity`s and `MergedEntity`s are added to the model. The *unit*s of all elements are collected and their redundancy is removed by a comparison of each of the units. They are then added to the model. In a final step the function returns the newly created SBML model.

3.3.3 Merge GUI

The GUI of the new merge algorithm is basically a table in which the rows represent elements and the columns represent models. The first column of the table represents the merged model. All other remaining columns represent the models that should be merged. Each row stands for one element of the merged model (which is about to be created). If there are matching elements their merged element is shown in the first column. If an element of a model does not match

any other element the first column is left blank. In the creation of the merged model the non matching element will be copied (with some modifications) into the merged model.

The following table shows an example in which two models (ModelA and ModelB) are merged. Each model has four elements. Two elements from each model are matching an element of the other model (ModelA-ATP with ModelB-ATP and ModelA-Glucose with ModelB-gluc) and two elements in each model are not matching any other element (ModelA-H1O, ModelA-Ethanol, ModelB-F6P, ModelB-F16bP).

Merged Model	ModelA	ModelB
Merged ATP	ATP	ATP
Merged Glucose	Glucose	gluc
	H2O	
	Ethanol	
		F6P
		F16bP

The main widget of the merge GUI is a toolbox widget. The toolbox widget consists of a list of vertically arranged tabs that extend when the tab header is clicked on. Inside a tab widgets that each display an element are arranged horizontally. The Figure 25 shows a screenshot of the merge GUI. The screenshot is composed of four separate screenshots which were combined for a better overview.

Legend of the Figure 25

1. The header is showing which model is located in which column.
2. Symbols representing the conflict status of the element.
3. Header of the element displaying the element name. The header is color coded for the different types of libSBML elements. In this case the blue highlighting represents a SBML *species*.
4. Drop-down-box widget that contains a list of similar elements with which the element could also be merged. Just below it a similar drop-down-box widget is located with a list of all elements that this element could be merged with. The lists only contain elements that do not belong to the same model.
5. Push button to remove the element from its current tuple. In this case pushing the button destroys the tuple and deletes the merged element.
6. Aggregated MIRIAM annotation with hyperlink.
7. Push button to delete the MIRIAM annotation. In case the aggregated MIRIAM annotations do not represent the merged element.
8. Unit string representation.

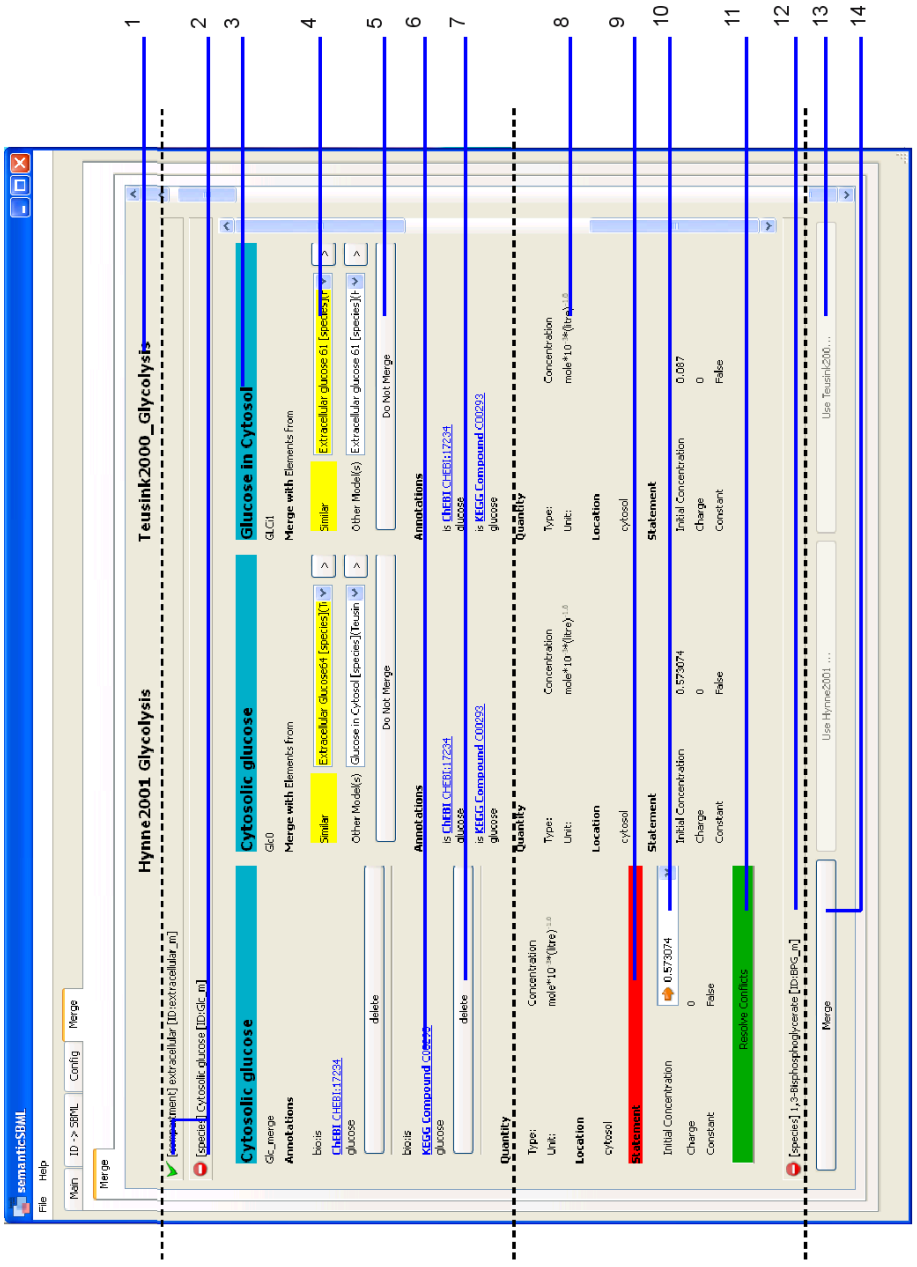

Figure 25: The figures shows a screenshot of the the new merge algorithm GUI. The screenshot is composed of four screenshots that were combined to present a better overview. The dashed lines indicate the borders of the single screenshots. The legend to this Figure can be found in Section 3.3.3.

9. Header of the model statement section. The red highlighting indicates that there are conflicting values in this section. When the resolution of the conflict was successful, it will be highlighted in green.

10. Editable drop-down-box widget that contains choices for the conflicting values. The current value is indicated by an arrow icon.

11. Resolve push-button which will execute the resolution functions. Upon pressing the button a pop-up window can appear that displays a severe conflict that prevents the resolution of the conflicts.

12. Tab header of the next element (tuple). Clicking the header will open this tab and close the current one.

13. Disabled push-button that is planned to execute a resolution of all conflicts by choosing only values of one model. This feature is still under development.

14. Merge push-button - if all conflicts are resolved pushing this button will create the SBML model and destroy the view. If there are still elements with unresolved conflicts a pop-up will appear that shows a list of these elements.

3.3.4 Discussion

The merging in semanticSBML allowes the user more freedoms than SBMLmerge while trying to prevent errors that result from these freedoms. New features are the merging of different element types and the manual choice of elements that should be merged. It includes methods for a simple recognition of severe conflicts and semantic checks of the user selected conflict resolution. The representation of the merge process is more transparent due to the merging of an arbitrarily number of models at once and an improved visualization of element values.

In the current state the program delivers a framework that needs further improvement. Missing features for example are the resolution of circular *rule* definitions (which was included in SBMLmerge) and circular *compartment* inclusions. Further semantic checks are needed which can only be discovered with a deeper understanding of the SBML format. The program does not yet support all SBML element types.

An improved implementation of the resolution of the dependency problem (in which only the depending elements are recreated) was started but could not be finished with this work.

SemanticSBML currently compares even complex elements like mathematical statements and units by string comparison. While this might seem ineffective experience showed that it yields some success (especially if standard units are used and models are derived from each other). An appropriate solution for the units comparison would be a conversion of the units or a standardization of units before the comparison. The libSBML developers announced that functions for a unit conversion are currently integrated in the libSBML and will be an inherent

part in future version of the library. In a discussion with the developers of COPASI [35] it was discovered that the comparison of mathematical statements is also needed in COPASI. It was agreed that the source code for the comparison of mathematical statements would be provided by the COPASI developers.

The identification of duplicate objects consits of two steps. The first step is the recognition of identity and the second is the generation of a similarity score. The similarity score is generated by a rather naïve algorithm. It uses a score matrix to enable a comparison of annotations with different qualifiers (e.g., annotation of model1: ATP "is version of" URI1#ID1 - compared to annotation of model2 ATP "is" URI1#ID1). The score matrix was intentionally not included in this thesis, since no combination of qualifiers other than "is" for both annotations was found to be more meaningfull than other combinations (and has a higher value in the score matrix). It can be argued that only elements with identical annotations and the qualifier "is" should be recognized as equal. This argument however would not allow elements that only have *weak annotations* (e.g., "is version of") to match any other element. The assumption made in the implementation is that if the argument would be would be true every biological entity can be referenced with an "is" qualifier. However since no database is complete the argument can not be valid. The implementation has the disadvantage that many weak annotations can outvote a strong annotation. Taking the BioModels database as the primary source of MIRIAM annotated models an outvoting in randomly merged models was not observed. With more sources for MIRIAM annotated SBML models, the current solution might turn out to be insufficient and has to be revised. On this account the matching function was placed into the `BioModels` class which was created on purpose as an extra class to contain critical algorithms separate from other algorithms.

The severity of conflicting `BioEntity` and `BioQuantity` attributes is much lager than of that of `ModelStatement` attributes. A user must be properly instructed to understand this in a public released version of semanticSBML. A severe conflict can mean that a merging of certain entities will result in the creation of a faulty model. Since the user can manually decide which entities should be merged it was decided that a warning of severe conflicts is a better solution than to prevent the merging of entities with sever conflicts.

4 Experiments

The following experiments were conducted to prove the functional efficiency of the new algorithms as well as to evaluate the usefulness of the concepts used in semanticSBML. The experiments also show alternative uses of semanticSBML. All models used in the experiments originate from the current release of the curated BioModels database (25th September 2007 release http://www.ebi.ac.uk/biomodels/).

4.1 Clustering

The MIRIAM annotation in SBML allows an automated identification of biological entities. A SBML model can be defined by the biological entities it uses to describe a biological phenomenon. These two facts were combined to create an automated method of finding similar models.

The datasource of the experiment was the complete BioModels database (25th

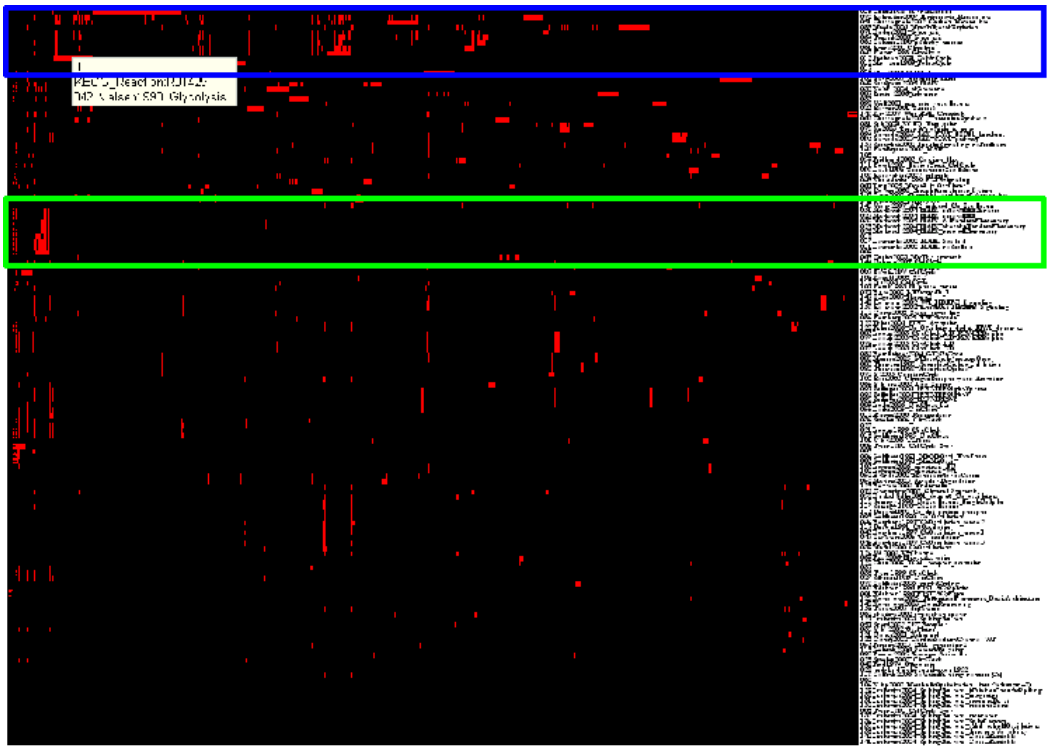

Figure 26: Clustered BioModels database, the rows represent models and the columns MIRIAM annotations of model elements. Two clusters are outlined. An enlarged view of the blue outlined cluster can be found in Figure 27 and a green outlined cluster in Figure 28.

September release, curated models only). The semanticSBML annotation API was used to extract the MIRIAM annotations from each model. A Python script was written that generates a matrix of the occurrence of MIRIAM annotations

in each model. The matrix contains a `1` if an identifier occurs in a model and a `0` if not. The qualifiers of the annotations were ignored. This matrix was then loaded into MATLAB [36] and clustered with the `clustergram` function which uses a hierarchical clustering algorithm. The clustered matrix was visualized and manually analyzed.

Multiple groups of models were recognized. Figure 26 shows an overview of the complete database with two regions outlined in blue and green. In the outlined regions dense red areas can be recognized. The areas represent clusters of MIRIAM annotations and models. The best observable cluster (blue outline, enlarged image in Figure 27) contains 5 models that describe the glycolysis (BioModel 42 [37], 61 [13], 64 [12], 71 [38], 63 [39]). The second cluster that can easily be recognized (green outline, enlarged image in Figure 28) contains 9 models that describe the reaction networks of the mitogen-activated protein kinase (BioModel 9 [40], 11 [41], 14 [41], 10 [42],26 [43],28 [43],30 [43],27 [43] ,31 [43]). 5 of these 9 models originate from the same work.

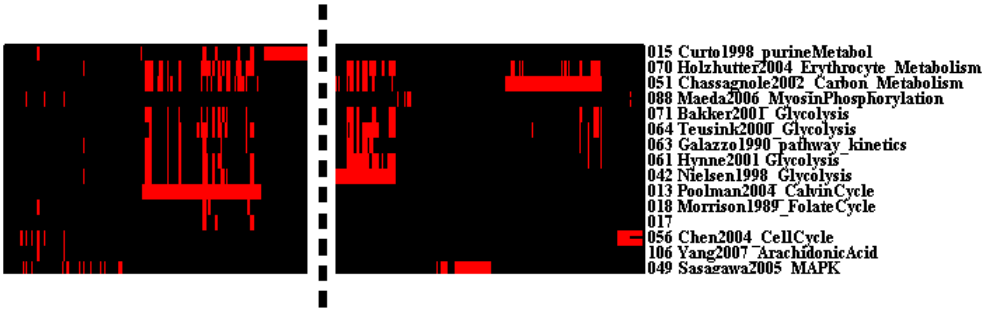

Figure 27: Close-up of a cluster of models that describe the glycolysis and similar biochemical networks. The dashed line indicates that part of the image was removed.

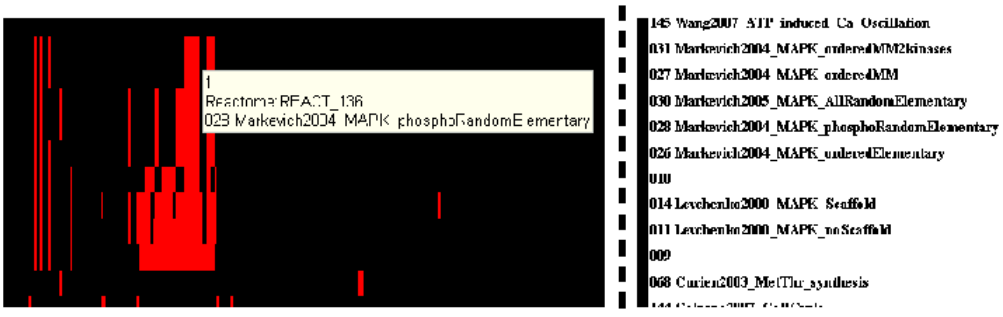

Figure 28: Close-up of a cluster of models that describe reaction networks around the mitogen-activated protein kinase. The dashed line indicates that part of the image was removed.

The clustering is a demonstration of an alternative usage of the semanticSBML annotation API. The experiment was conducted with rather vague assumptions. To achieve better results only annotations with the qualifier "is" should be used. Furthermore it is recommended to choose a more suiting clustering algorithm. However even with these loose assumptions the clustering shows the potential of an easy access to MIRIAM annotations. The clusters that were presented in this experiment could easily be found by a manual comparison, but if the amount of available models rises a manual search for similar models could become a very time consuming task. The clustering of models deliver a great alternative to the manual search for similar models.

4.2 Analysis of Merging Two Glycolysis Models

The example in the introduction Section 1 shows two models that described similar aspects of the glycolysis of Saccharomyces cerevisiae. In this experiment it is attempted to merge the glycolysis model of Hynne *et al.* [13] (BioModel 61) and the glycolysis model of Teusink *et al.* [12] (BioModel 64) with semanticSBML. The merging process will be analyzed and discussed.

The merging starts with the comparison of elements and a creation of a randomly merged model (see Section 3.3.2). The following table shows the result of the comparison by listing the number of mergable elements by type for each model and the number of elements that were recognized as duplicates.

Element Type	Teusink Model	Hynne Model	Matching Elements
compartment	2	2	2
species	21	25	13
reaction	18	26	10
parameter	21	0	0

The first thing that can be recognized is that all matching elements have conflicting values.

Both *compartment*s in each model were matched but contain conflicts in their annotations. The annotation *resource* (URI and Identifier) in these annotation are recognized as identical. However in the Hynne model the MIRIAM annotations have the qualifier "is version of" while in the Teusink model annotates the elements with the qualifier "is". A matching of the elements was possible since the merging algorithm allows the matching of elements with *weak* annotations (see Section 3.3.2 and 3.3.4). The option is given to chosen which qualifier each of the annotations should have in the merged model. The GUI marked the qualifier "is" as preselected and since this qualifier was chooses it was only needed to press the "resolve" push-button.

Out of the 13 matching *species* 11 of the *species* had conflicting initial concentrations. No other conflicts occurred in these 11 *species*. The conflicts were resolved by choosing the preselected concentrations. The two remaining matching elements had next to a conflicting initial concentration also a conflict in their annotations.

The *species* "High energy phosphates" (the element names are used in this description) of the Teusink model matched the *species* "ATP" in the Hynne model. The analysis of the annotation conflict revealed that the element of the Teusink model had two resource (KEGG compound identifier C00002 and ChEBI identifier ChEBI:15422) with the qualifiers "has part" that were identical to the resources in the Teusink model with the qualifiers "is". On checking the Teusink model a species "P" was found that would biologically match the species "High energy phosphates" in the Hynne model. However the "P" species was not MIRIAM annotated and was thus not recognized as a match. This problem could be resolved by manually matching the two biologically matching *species*. In the current state of semanticSBML the manual matching of elements does not update depending elements and thus excluded this solution. As an alternative solution a user could add the correct annotations to the Hynne model species "P" before the merging is performed. This conflict resolution bypasses the *species* "ATP" since no similar species could be found in the Teusink model. This is a hint that there are more severe problems between the concepts of the two models.

The second pair of matching elements with conflicting annotations were the elements "Triose-phosphate" (Teusink model) and "Glyceraldehyde 3-phosphate" (Hynne model). Similar to the problem in the paragraph above the Teusink model had two annotations with the qualifier "has part" matching the annotations with the qualifier "is" found in the Hynne model. In addition to that the *species* "Triose-phosphate" showed that there was a second matching element in the Hynne model "Dihydroxyacetone phosphate" (the generation of a list of similar elements was described in Section 3.3.2). The hyperlinked external resources reveal that the two *species* matching the "Triose-phosphate" are very similar, and a further investigation in the linked resources show that there is a reaction that converts the chemical compounds into one another. In the analysis of the reactions of the models it shows that the conversion reaction is part of the Hynne model but absent in the Teusink model. In the current version of semanticSBML this problem is very difficult to solve since an aggregation / splitting of elements is not possible.

The *species* that were not matched were analyzed. It was found that at least two *species* were not recognized as matching due a problem in their annotations. The species "Glucose 6 Phosphate" and "Fructose 6 Phosphate" (Teusink model) should have matched with the species "Glucose-6-Phosphate" and "Fructose-6-Phosphate" (Hynne model) however the match was missed because the Teusink model annotated the elements with α-D-glucose 6-phosphate (ChEBI identifier ChEBI:17665 and KEGG Compound identifier C00668) and β-D-Fructose 6-phosphate (KEGG Compound identifier C05345) respectively while in Hynne model the annotations referenced D-glucose 6-phosphate (ChEBI identifier ChEBI:15954 and KEGG Compound identifier C00092) and D-Fructose 6-phosphate (KEGG Compound identifier C00085). This problem could be solved by extending the internal database to provide extended information of object relations (in this case one object is the parent of the other object).

All matching *reaction*s had conflicting *kinetic law*s. The choice of the *kinetic law* needed a deeper understanding of the models and therefore the the preselected values were chosen.

Six of the ten matching reactions had conflicting reaction participants (reactands, products). In four cases the participants were conflicting in the inclusion and exclusion of ATP/ADP/P as reaction participant. In one case the already mentioned problem of the splitting of the "Triose phosphate" *species* in the Teusink model into two *species* in the Hynne model can be seen again. The reaction "Aldolase" (same name for both models) has in the Teusink model one product and in the Hynne model two products. The reaction participant conflict of the matching "Glucose-6-Phosphate isomerase" (Teusink model) "Phosphoglucoisomerase" (Hynne model) elements hints that the comparison step missed biologically matching elements (in this case glucose-6-phosphate and fructose-6-phosphate as discussed above). The further analysis of non matching *reaction*s revealed similar problems as the ones already mentioned.

It should be noted that there were no conflicts between units.

The analysis of the merging shows that a manual matching process is inevitable. Furthermore it can be seen that an aggregation of elements is an interesting field and that semanticSBML needs further improvement. The matching of weak annotations was proven to be usefull but it could also be seen that weak matches deserve special attention in the merging process. The creation of the randomly merged model revealed missed matches and thus also proved to be usefull. The merging of the two models in this experiment might not be possible in the current state of semanticSBML but it showed the strength and weaknesses. Experiments like this will help in the improvement of semanticSBML and possibly also other tools currently in development.

4.3 Merging of Respiratory Oscillation Model

To verify the functional efficiency of the new merge algorithm (see Section 3.3) a model created by Wolf *et al.* [21] (BioModel 90) was used. The model constits of an oscillating reaction network within a cell which is powered by two extracellular substances. The model was merged with itself in a way that the cell was duplicated while entities in the extracellular space were merged. In a successful merging of the oscillation of the cell should be seen in both cells with identical concentrations for each substance. A simulation was performed with COPASI 4.2.23 (development) and the results were analyzed.

The model was prepared by creating a copy of the original model and renaming the copied model. Since the manual modification of dependant elements is currently not possible in semanticSBML the models were prepared in such a way that the compare step (see Section 3.3.2) would directly yield the desired matches. On this account all MIRIAM annotations except for those of extracellular entities (including the *compartment* "extracellular space" itself) of the copied model were removed. The removing of the annotations was done using the semanticSBML annotation algorithm. The original model and the copy was

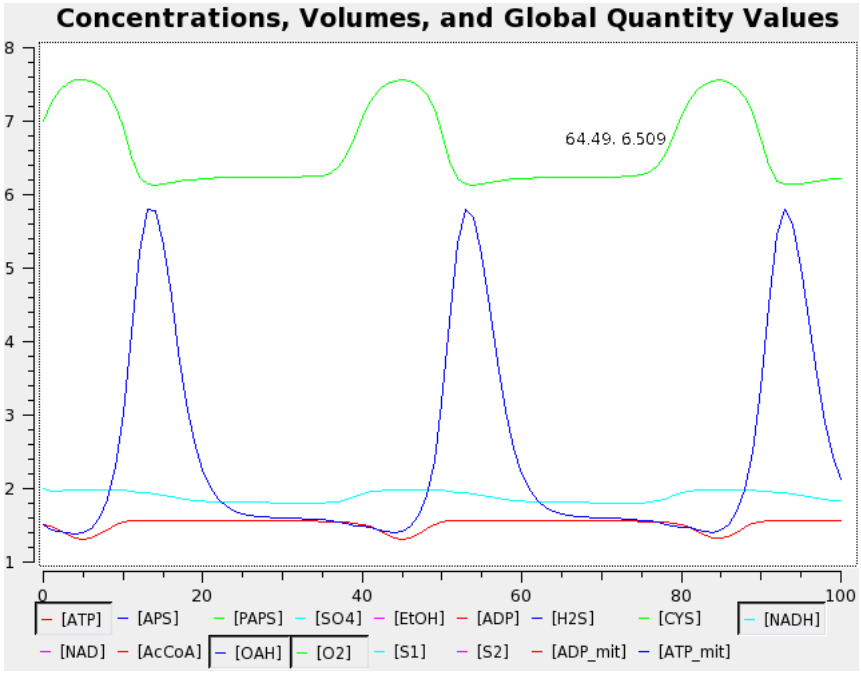

Figure 29: Simulation of the "Respiratory Oscillation Model" (Wolf *et al.* 2001) in COPASI with four representative *species* that show oscillating concentrations over time.

then merged. Since the models were identical (except for their MIRIAM annotations) there were no conflicts in the matching elements and the model could be merged directly.

The original model and the merged model were loaded successfully into COPASI and a *time course* simulation was conducted for 100 steps. The concentrations values of the *species* were plotted against time. Figure 29 shows the result of the simulation for the original model. Four non extracellular species were selected that showed oscillating concentration values. The Figures 30 and 31 show the results of the simulation of the merged model. The same species were selected to show the oscillation in the merged model. The concentration values of each species in one of the cell compartments were exactly the same. The concentration values in the merged model however differed from those in the original model. This is due to the fact that the substance powering the reactions in the cell were used up by two cells in the merged model. This means that only half of the substance amount can be used by each cell.

The experiment proves the functional efficiency of the new merging algorithm to create complete and simulatable models. The created model has little scientific value in itself but it shows an alternative usage of semanticSBML than merging different models into one model. Even though the merging algorithm is still incomplete, a successful merging could be conducted using only functions of semanticSBML.

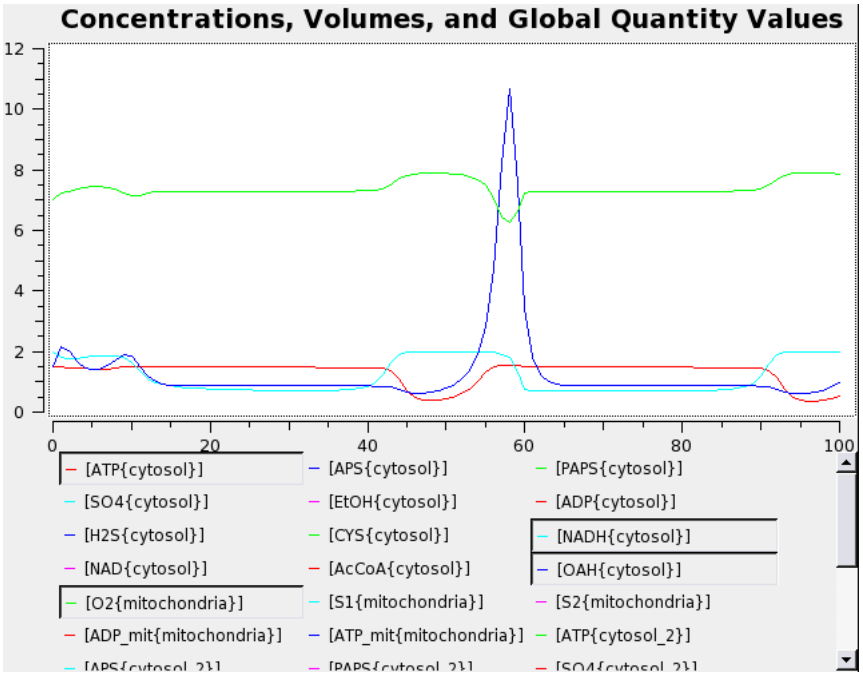

Figure 30: Simulation of the merged respiratory oscillation model. Four oscillating concentrations of selected species are shown. The values are identical to those of the identical species in the second cell (see Figure 31).

5 Conclusion

In the first phase of the development a fully functional release of semanticSBML was created. The release includes a clean installation of semanticSBML. The GUI was completed by adding the creation and merging of SBML models functions (Section 2.3). A CI was developed which includes a batch processing ability that enables a user to automate the functions of semanticSBML (Section 2.4). In addition to that a simplified API was introduced that enables the usage of semanticSBML as a external programming library (Section 2.2). The functions of semanticSBML in the first development phase are based on SBMLmerge. In Section 2.3.2 the GUI to the SBMLmerge merging algorithm was introduced and the problems of the old merge algorithm were shown. In the second development phase this information was used for the creation of a new merging algorithm.

The SBML MIRIAM annotation plays and important role in the merging of SBML models and in the creation of models that should be released to the public. On this account the MIRIAM annotation manipulation algorithm was updated to fit the current status (Section 3.2). The update was achieved by a complete rewrite of the annotation algorithms since the underlying library libSBML added a native support for MIRIAM annotations as well as for the fact that a more flexible design could be achieved. The new annotation algorithms also introduced a new API and an improved GUI for the manipulation

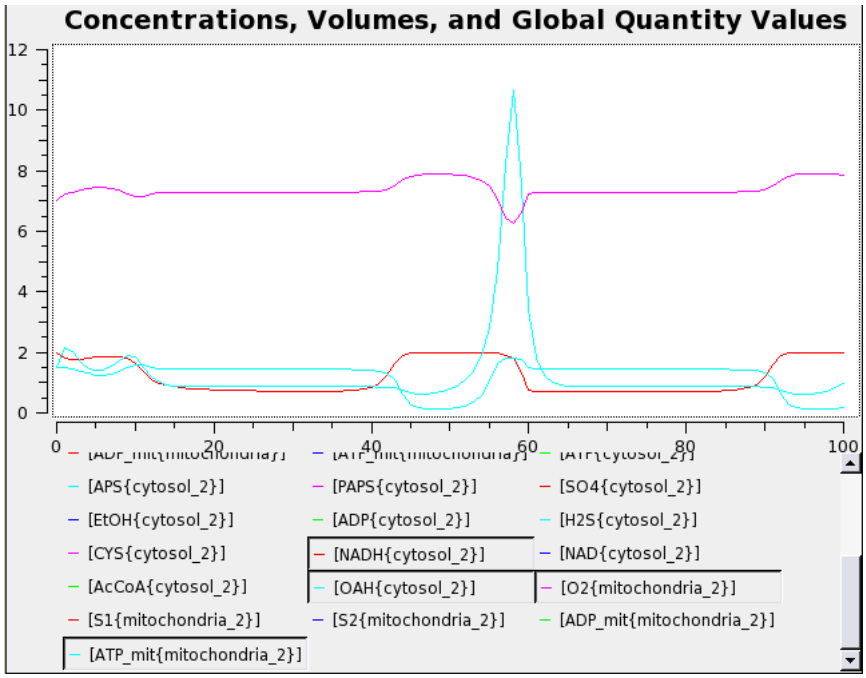

Figure 31: Simulation of the merged respiratory oscillation model. Four oscillating concentrations of selected species are shown. The values are identical to those of the identical species in the first cell (see Figure 30).

of MIRIAM annotations. In an experiment the new annotation API was used to cluster similar models in the BioModels database (Section 4.1).

Just like the annotation algorithm the merging algorithm was rewritten (Section 3.3). A datastructure for the merging was created that has in its center semanticSBMLs own abstraction of a systems biology model. For a number of problems strategies were developed to enable a safe and userfriendly merging of models. While the merging algorithm is not in a state that it can be released to the public, it does deliver a strong framework that can be used to create function merged models. Its problem resolution strategies were analyzed in an experiment (Section 4.2) and in another experiment (Section 4.3) its functionality was proven.

6 Further Work

The development of semanticSBML is not complete. The next important step is the development of a computational lightweight method to update elements after a manual manipulation of the model structure. The merging algorithm has to be tested intensely and remaining problems have to be eliminated. The testing will mostlikely reveal problems in the semantics of the created models that have to be prevented by integrating further semantic checks. The GUI of the merge algorithm has a couple of small problems e.g. missing scrollbars that have to be corrected in order to provide a pleasant merging experience with semanticSBML. The CI delivers a method of automating the merging process. The CI is currently not working with the new merge algorithm. In the updating process of the CI it is aspired to create a method for the documentation of the merging process. This means that all operations that were conducted during the merging process should be protocolled so that the merging process can be repeated.

As it was mentioned in the discussion of the new annotation algorithm (Section 3.2.7) the internal database needs to be restructured. In the current state the functions of the internal database to retrieve information of the identity of different identifiers needs improvement. As it was shown in the experiment in Section 4.2 semantic information of identifier relations can improve the merging process. String representations of annotations are a good aide in the identification of entities. An extension of the amount of data (while retaining a fast data access) would help the identification of entities by a user and thus improve the the usage of semanticSBML.

The semanticSBML project is now in its third year of development and has improved the program greatly. It is my hope that the development can continue in the future to create a software that will aide the systems biology community to better understand biological life.

A Frequently used Terms

The following list explains special terms that are used in this thesis.

dictionary
: Hashtable implementation in the Python programming language.

module
: A module represents a file in Python. A module usually contains classes but it can also contain functions directly.

function
: A subroutine, also known as method, procedure, or subprogram.

`__init__` / `__new__`
: A special method used to initialize classes, similar to a constructor in other programming languages

biological entity
: A biological object e.g., the chemical compound ATP, the compartment cytosol, the reaction that converts Glucose into Glucose-6-phosphate.

qualifier
: A qualifier defines the relationship (e.g., "has part", "is version of") between two objects e.g., the relationship between a biological entity (ATP) and a database entry (Reactome entry for ATP). A detailed description of qualifiers can be found in Section 3.2.1.

widget
: A graphical element e.g., a push putton.

to patch
: Correcting of a flawed algorithm.

to wrap
: Creating a function that has the same input and output of another function.

to raise
: Raising an exception is also known as throwing an exception.

signal - slot
: Qt concept of executing functions on (user generated) events e.g., the execution of the function that creates a pop-up window on the pressing of a push-button.

to port
: Updating of interface functions of an integrated library. For example the function `setCaption` in Qt3/PyPt3 was renamed to `setWindowTitle` in Qt4/PyQt3.

serialization / deserialization
: In the context of this thesis it is the process of writing an object / class instance to the harddrive (serialization) and then creating the object / class instance again from the file on the harddrive (deserialization).

B SBML base elements

The SBML format is a hierarchic format that has in its first level a row of element types that represent the main concepts of the SBML model. These elements are referred to as *base elements* or *libSBML elements* in this thesis. In libSBML the base elements can be accessed from the model instance with the `listOf*` functions. In this thesis the SBML elements are written in a *slanted font*. Since the names of the elements also represent biological concepts the element and the concept can not always be differentiated. The following description gives and overview of the most important SBML base elements used in this thesis. A full description can be found in the SBML specification [10].

species
: The *species* element represents a physical entity e.g., a chemical compound like ATP or a protein or protein complex.

compartment
: The *compartment* represents a bounded space in which *species* are located e.g. the nucleus, the cellwall or the cytosol

reaction
: The *reaction* element represents a transformation, transport or binding process, typically a chemical reaction, that can change the quantity of one or more *species*. The *reaction* contains a mathematical statement (the *kinetic law*).

parameter
: Not all symbols used in mathematical statements in SBML must be defined by e.g. *species*, *compartment*. The *parameter* defines a symbol that is associated with a values. However in different models a parameter can be represented by a *species* or *compartment* and vice versa.

initial assignment
: An assignment of an initial value of an entity (e.g., *species*, *compartment*, *parameter*) can be archived by either setting the `value` attribute of the element or by using a mathematical expression (the *initial assignment*). An *initial assignment* always refers to another element.

rule
: The libSBML element *rule* contains a mathematical statement that is used to define dynamic properties of an entities (e.g., *species*, *compartment*, libSBML *parameter*) value. There are three different types of rules: *algebraic rule*, *rate rule* and *assignment rule*. The *rate rule* and *assignment rule* always refer to another element. The *algebraic rule* represents a mathematical statement that has the general from $0 = f(x)$ and can thus refer to many elements.

References

[1] Snoep, J. L., Bruggeman, F., Olivier, B. G. & Westerhoff, H. V. Towards building the silicon cell: a modular approach. *Biosystems* **83**, 207–216 (2006). URL http://dx.doi.org/10.1016/j.biosystems.2005.07.006.

[2] Nielsen, P. F. & Halstead, M. D. The evolution of CellML. *Conf Proc IEEE Eng Med Biol Soc* **7**, 5411–5414 (2004). URL http://dx.doi.org/10.1109/IEMBS.2004.1404512.

[3] Hermjakob, H. *et al.* The HUPO PSI's molecular interaction format–a community standard for the representation of protein interaction data. *Nat Biotechnol* **22**, 177–183 (2004). URL http://dx.doi.org/10.1038/nbt926.

[4] Luciano, J. S. PAX of mind for pathway researchers. *Drug Discov Today* **10**, 937–942 (2005). URL http://dx.doi.org/10.1016/S1359-6446(05)03501-4.

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

[6] Strmbck, L. & Lambrix, P. Representations of molecular pathways: an evaluation of sbml, psi mi and biopax. *Bioinformatics* **21**, 4401–4407 (2005). URL http://dx.doi.org/10.1093/bioinformatics/bti718.

[7] Home site for the systems biology markup language (sbml) (2007). URL http://sbml.org/ (retrieved December 2007).

[8] Extensible markup language (xml). URL http://www.w3.org/XML/.

[9] Finney, A. & Hucka, M. Systems biology markup language: Level 2 and beyond. *Biochem Soc Trans* **31**, 1472–1473 (2003). URL http://dx.doi.org/10.1042/.

[10] Hucka, M., Finney, A., Hoops, S., Keating, S. & Novre, N. L. Systems biology markup language (SBML) level 2: Structures and facilities for model definitions. http://sbml.org/specifications/sbml-level-2/version-3/release-2/sbml-level-2-version-3-rel-2.pdf.

[11] Novre, N. L. *et al.* Minimum information requested in the annotation of biochemical models (miriam). *Nat Biotechnol* **23**, 1509–1515 (2005). URL http://dx.doi.org/10.1038/nbt1156.

[12] Teusink, B. *et al.* Can yeast glycolysis be understood in terms of in vitro kinetics of the constituent enzymes? testing biochemistry. *Eur J Biochem* **267**, 5313–5329 (2000).

[13] Hynne, F., Dan, S. & Srensen, P. G. Full-scale model of glycolysis in saccharomyces cerevisiae. *Biophys Chem* **94**, 121–163 (2001).

[14] Schulz, M., Uhlendorf, J., Klipp, E. & Liebermeister, W. SBMLmerge, a system for combining biochemical network models. *Genome Inform* **17**, 62–71 (2006).

[15] Sanner, M. F. Python: a programming language for software integration and development. *J Mol Graph Model* **17**, 57–61 (1999).

[16] Trolltech. Qt: Cross-platform rich client development framework. URL `http://trolltech.com/products/qt/`.

[17] Karp, P. D. *et al.* Expansion of the BioCyc collection of pathway/genome databases to 160 genomes. *Nucleic Acids Res* **33**, 6083–6089 (2005). URL `http://dx.doi.org/10.1093/nar/gki892`.

[18] Vastrik, I. *et al.* Reactome: a knowledge base of biologic pathways and processes. *Genome Biol* **8**, R39 (2007). URL `http://dx.doi.org/10.1186/gb-2007-8-3-r39`.

[19] Snoep, J. L. & Olivier, B. G. JWS online cellular systems modelling and microbiology. *Microbiology* **149**, 3045–3047 (2003).

[20] Novre, N. L. *et al.* BioModels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Res* **34**, D689–D691 (2006). URL `http://dx.doi.org/10.1093/nar/gkj092`.

[21] Wolf, J., Sohn, H., Heinrich, R. & Kuriyama, H. Mathematical analysis of a mechanism for autonomous metabolic oscillations in continuous culture of saccharomyces cerevisiae. *FEBS Lett* **499**, 230–234 (2001).

[22] Riverbank. PyQt: Python bindings for trolltech's qt application framework. URL `http://www.riverbankcomputing.co.uk/pyqt/`.

[23] Epydoc: Automatic API documentation generation for python. URL `http://epydoc.sourceforge.net/`.

[24] Kanehisa, M. The KEGG database. *Novartis Found Symp* **247**, 91–101; discussion 101–3, 119–28, 244–52 (2002).

[25] Ashburner, M. *et al.* Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nat Genet* **25**, 25–29 (2000). URL `http://dx.doi.org/10.1038/75556`.

[26] Easy-Software-Products. EPM: ESP package manager. URL `http://www.epmhome.org/`.

[27] Funahashi, A. & Kitano, H. CellDesigner: a process diagram editor for gene-regulatory and biochemical networks. *BioSilico* **1**, 159162 (2003).

[28] RDF/XML syntax specification (revised) (2004). URL `http://www.w3.org/TR/rdf-syntax-grammar/`.

[29] Berners-Lee, T., Fielding, R. & Masinter, L. Uniform resource identifier (URI): Generic syntax. URL `http://www.gbiv.com/protocols/uri/rfc/rfc3986.html`.

[30] Berners-Lee, T. Uniform resource locators (URL): a syntax for the expression of access information of objects on the network. URL `http://www.w3.org/Addressing/URL/url-spec.txt`.

[31] The BioModels qualifiers. URL http://www.biomodels.net/index.php?s=Qualifiers (retrieved December 2007).

[32] Degtyarenko, K. *et al.* ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic Acids Res* **36**, D344–D350 (2008). URL http://dx.doi.org/10.1093/nar/gkm791.

[33] Huffenberger, M. A. & Wigington, R. L. Chemical abstracts service approach to management of large data bases. *J Chem Inf Comput Sci* **15**, 43–47 (1975).

[34] 3DMET: A database of three-dimensional structures of natural metabolites. URL http://www.3dmet.dna.affrc.go.jp/.

[35] Hoops, S. *et al.* COPASI–a COmplex PAthway SImulator. *Bioinformatics* **22**, 3067–3074 (2006). URL http://dx.doi.org/10.1093/bioinformatics/btl485.

[36] The MathWorks. MATLAB r2007a. URL http://www.mathworks.com/products/matlab/.

[37] Nielsen, K., Srensen, P. G., Hynne, F. & Busse, H. G. Sustained oscillations in glycolysis: an experimental and theoretical study of chaotic and complex periodic behavior and of quenching of simple oscillations. *Biophys Chem* **72**, 49–62 (1998).

[38] Helfert, S., Estvez, A. M., Bakker, B., Michels, P. & Clayton, C. Roles of triosephosphate isomerase and aerobic metabolism in trypanosoma brucei. *Biochem J* **357**, 117–125 (2001).

[39] Galazzo, J. L. & Bailey, J. E. Fermentation pathway kinetics and metabolic flux control in suspended and immobilized saccharomyces cerevisiae. *Enzyme and Microbial Technology* **12**, 162–172 (1990).

[40] Huang, C. Y. & Ferrell, J. E. Ultrasensitivity in the mitogen-activated protein kinase cascade. *Proc Natl Acad Sci U S A* **93**, 10078–10083 (1996).

[41] Levchenko, A., Bruck, J. & Sternberg, P. W. Scaffold proteins may biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties. *Proc Natl Acad Sci U S A* **97**, 5818–5823 (2000).

[42] Kholodenko, B. N. Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. *Eur J Biochem* **267**, 1583–1588 (2000).

[43] Markevich, N. I., Hoek, J. B. & Kholodenko, B. N. Signaling switches and bistability arising from multisite phosphorylation in protein kinase cascades. *J Cell Biol* **164**, 353–359 (2004). URL http://dx.doi.org/10.1083/jcb.200308060.