MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

System for generating visualizations of SBML models

DIPLOMA THESIS

Karol Pál

Brno, Spring 2011

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: Ing. Matej Lexa Ph.D.

Acknowledgement

I would like to thank my supervisor Ing. Matej Lexa Ph.D. for his guidance during the work on this thesis and during the development SBML Visualization program.

Abstract

The aim of this thesis is to design a program for SBML document visualization.

The first part of this work familiarizes the user with the concepts of SBML language, Gene Ontology and explains the problem that should be solved. Next the technologies and ideas required for this solution are presented.

The second part introduces a functional program build on the proposed principals. Shows its work and explains how it is to be used.

Keywords

SBML,Gene Ontology, Graph visualization, Graph layout algorithm, JUNG library, SVG

Contents

Chapter 1

Introduction

The problematics of Bioinformatics and Systems Biology are in no case new and since they usually work with large amount of data they produce computationally difficult problems. Programs developed in this area tend to employ minimum amount of necessary information[1] to minimize their workload. As a result a lot of machine generated data acquired from such programs lacks a certain human dimension.

This thesis works with Systems Biology Markup Language (SBML) which is a standard format used for exchange of information between many programs of Systems Biology. The aim of this thesis is to provide a visualization for this machine generated code that would add some human comprehensible descriptive value to it.

The following chapter (Chapter 2) provides an introduction to the SBML language. Gives a background on the basic concepts of the language, justifies its use, and describes its structure. The chapter also familiarizes the reader with the concept of Gene Ontology which is also a key concept for this work.

The third chapter presents an insight into current available solutions to the problematic of SBML visualization, identifies their weaknesses and gives a proposal on how to make a program eliminating these weaknesses.

Chapter four describes the results of the process of implementation carried out as proposed in chapter three, analyses some pieces of source code, presents the working program and describes the user interface.

Finally chapter five studies a particular model visualization, which is the resulting product of the developed program.

1. which is still a significant amount

Chapter 2

Theoretical Background

2.1 SBML

"Systems Biology Markup Language (SBML) is a free, open, machine readable, XML-based format for representing biochemical reaction networks. SBML is a software-independent language for describing models common to research in many areas of computational biology, including cell signaling pathways, metabolic pathways, gene regulation, and others" [3].

According to HUCKA et al. [2] SBML has become a de facto standard format for representing formal, quantitative and qualitative models at the level of biochemical reactions and regulatory networks.

While SBML does not aim to be a universal language for representing quantitative models, one of the great strengths of SBML is its independence on any existing software. At the same time a lot of software relies on SBML and many developers approach it as a "common intermediate format - a lingua franca - enabling communication of the most essential aspects of models"[2].

The need for such a language was expressed at the *Workshop on Software Platforms for Systems Biology*, held at the California Institute of Technology in early 2000, by a few dozen software projects[2] (Including Cellerator[6], E-Cell[7], Virtual-Cell[5]). As of today there are over 200 software packages listed in the SBML Software Matrix[1]. The list consists of tools for computational modeling in systems biology, software libraries for programming with SBML, interfaces to popular general-purpose mathematical environments, conversion tools and Online web-based facilities.

2.1.1 SBML Levels

The SBML development process is marked by *levels* and *versions*. Major editions of SBML are called levels[4]. The level hierarchy builds every higher

1. Available at :`http://sbml.org/SBML_Software_Guide/SBML_Software_Matrix`

level on top of the previous one. Each new higher level feature adds richness and expressiveness based on practical experience with the language definition. A higher level adds representation power but does not render previous levels obsolete [2]. Levels upon revision are released in versions. Versions of a level however do supersede each other.

As of today the most up to date level is SBML Level 3 Version 1 Core. However it does not yet seem to be in use and the newest models probably expecting future revisions of Level 3 are still supporting the most common SBML Level 2 Version 4.

2.1.2 SBML Structure

An SBML model consists of lists of one or more *compartments* which are containers of finite volume where reactions take place; biochemical entities (*species*): a pool of concentrations or amounts of chemical substances (not single molecules) interconnected by *reactions*. Reactions describe transitions, transport or binding processes characterizes by their products and reactants. Furthermore the list consists of *parameters*: quantities with symbolic names;*unit definitions* names for units used in the expression of quantities in a model; *rules*: mathematical expressions added to the equations constructed from the set of reactions; *functions*: named mathematical functions; *events*: sets of mathematical formulas evaluated at specified moments.

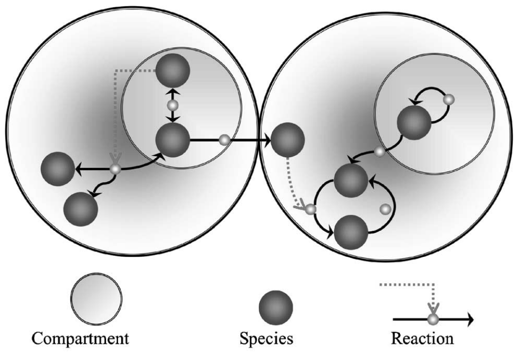


Figure 2.1: SBML model scheme

2. The compatibility is not straight forward. All of constructs of a lower level can be mapped to higher level constructs and in addition a subset of a higher level constructs can be mapped to lower level constructs. Nevertheless levels remain distinct. I.e. a valid SBML Level 1 document is not a valid SBML Level 2 document and vice versa.[4]

```
<model name="Example">
<listOfCompartments>
  <compartment name="Mithocondrial Matrix" id="MM"/>
</listOfCompartments>
<listOfSpecies>
  <species name="Succinate"  compartment="MM" id="Succinate" />
  <species name="Fumarate" compartment="MM" id="Fumarate" />
  <species name=”Succinate dehydrogenase"
           compartment="MM" id="Succdeh" />
</listOfSpecies>
<listOfReactions>
  <reaction name="Succinate dehydrogenas catalysis" id="R1">
    <listOfReactants>
      <speciesReference species="Succinate" />
    </listOfReactants>
    <listOfProducts>
      <speciesReference species="Fumarate" />
    </listOfProducts>
    <listOfModifiers>
      <modifierSpeciesReference species="Succdeh" />
      <modifierSpeciesReference species="S4" />
    </listOfModifiers>
  </reaction>
</listOfReactions>
</model>
```

Figure 2.2: SBML Example code

2.1.3 SBML alternatives

Of the many XML-based formats representing data and models in biology e.g. BIOpolymer Markup Language(BIOML),Chemical Markup Language (CML) Proteomics Experiment Markup Language (PEML)... HUCKA et al. [2] state that only two are suitable for representing compartmental reaction network models with sufficient mathematical depth. These are SBML and CellML[1].

CellML focuses on a component based architecture to facilitate reuse of models and parts of models. On the other hand SBML provides constructs that are more similar to the internal data objects used in many simulation/analysis software packages specialized for biochemical networks.

Both projects initially developed independently use different approaches to solve basically the same set of problems. The primary developers of both languages are actively engaged in exchange of ideas and are making the formats easier to translate between each other [2].

2.1.4 A note on MIRIAM

Upon examining the example of figure 2.2 code of SBML it is obvious that except the *id* and the *name* there is not much information regarding the entities in the lists. In practice the brevity of SBML is compensated by **notes** and **annotation** elements. While notes (most often used only in the `<model>` tag) contain human readable information often stored in XHTML format, the annotation elements are "containers for optional software-generated content *not* meant to be shown to humans." [4]. The element's content type is XML type **any** which allows essentially arbitrary well-formed XML data content. Nevertheless when referring to "controlled vocabulary terms and database identifiers which define and describe biological and biochemical entities" HUCKA et al.[4] recommend using a subset of the non-proprietary RDF format where external resources are referenced by a `MIRIAM_URN`. MIRIAM stands for Minimum Information Required in the Annotations of Models.

For example a MIRIAM resource identifier referring to a nicotinic acetylcholine-gated receptor-channel complex Gene Ontology entry looks like this: `urn:miriam:obo.go:GO%3A0005892` The list of data types stored in MIRIAM Database are available at: `http://www.ebi.ac.uk/miriam/main/datatypes/`

2.2 Gene Ontology

In the context of information science an **Ontology** is a common terminology used to describe data; or a "formal and explicit specification of *terms* used and the *relationships* between them"[8]. Such a unified vocabulary enabling sharing and reuse of knowledge is especially important when there are many groups of researchers working with essentially the same complex data.

The **Gene Ontology (GO)**[3] (part of The Open Biological and Biomedical Ontologies (OBO)[4] is a "collaborative project across many laboratories to provide a controlled vocabulary that describes gene and gene-associated

3. Home page: `http://www.geneontology.org`
4. Home page: `http://obofoundry.org/`

information (but not gene products) for all organisms"[8]. The GO does not include gene products; processes functions or components that are unique to mutants or diseases; gene sequences; protein domains or structural features.

2.2.1 Gene Ontology Structure

The Gene Ontology project consists of three distinct ontologies. The **Cellular Component** ontology describes locations, at the levels of subcellular structures and macromolecular complexes. A gene product is "located in" or is a "subcomponent of" a particular cellular component. The **Biological Process** ontology recognizes series of events or molecular functions. A process is a collection of molecular functions with a defined beginning and end. Finally the **Molecular function** ontology describes activities (rather than entities - molecules or complexes - that perform the action), such as catalytic or binding activities, that occur at the molecular level.

A **GO Term**, the basic building block of the ontology, consists of two required tags and optional tags. The required tags are `id` unique identifier of a term and the `name` of the term. Any term may have only one name defined (the name consists of a string of words).

The optional tags may assign any number of alternative ids (`alt_id`) to a term. The `namespace` tag value is the name of one of the three ontologies of GO where this tag belongs to. A term can contain (up to one) definition (the `def` tag) and comment (`comment`). The `subset` tag indicates a term subset to which this term belongs and the `synonym` tag gives a synonym for this term where the scope of a synonym may be one of four values: `EXACT, BROAD, NARROW, RELATED`.

Remaining tags specify relationships between terms. A term can contain an arbitrary number of the following tags: `is_a` describes a subclassing relationship between one term and another, the value being the id of the term of which this term is a subclass to; `intersection_of` indicates that this term is equivalent to the intersection of several other terms; `union_of` indicates that this term represents the union of several other terms; `disjoint_from` for terms that have no instances or subclasses in common. Any other relationship of terms can be represented in the `relationship` tag where the value of this tag should be the relationship type id an the id of the target term (for example the `part_of` relationship which is the most important of the relationships formed this way).An example Gene Ontology entry is shown in listing 2.1.

An effort is made to maintain complete `is_a` and `part_of` relation-

Listing 2.1: GO Term example

```
id: GO:0030314
name: junctional membrane complex
namespace: cellular_component
def: "Complex formed in muscle cells between
        the membrane of the sarcoplasmic reticulum
        and invaginations of the plasma membrane
        (T-tubules)." [PMID:11535622]
synonym: "triad junction" BROAD []
is_a: GO:0043234 ! protein complex
is_a: GO:0044444 ! cytoplasmic part
relationship: part_of GO:0016528 ! sarcoplasm
```

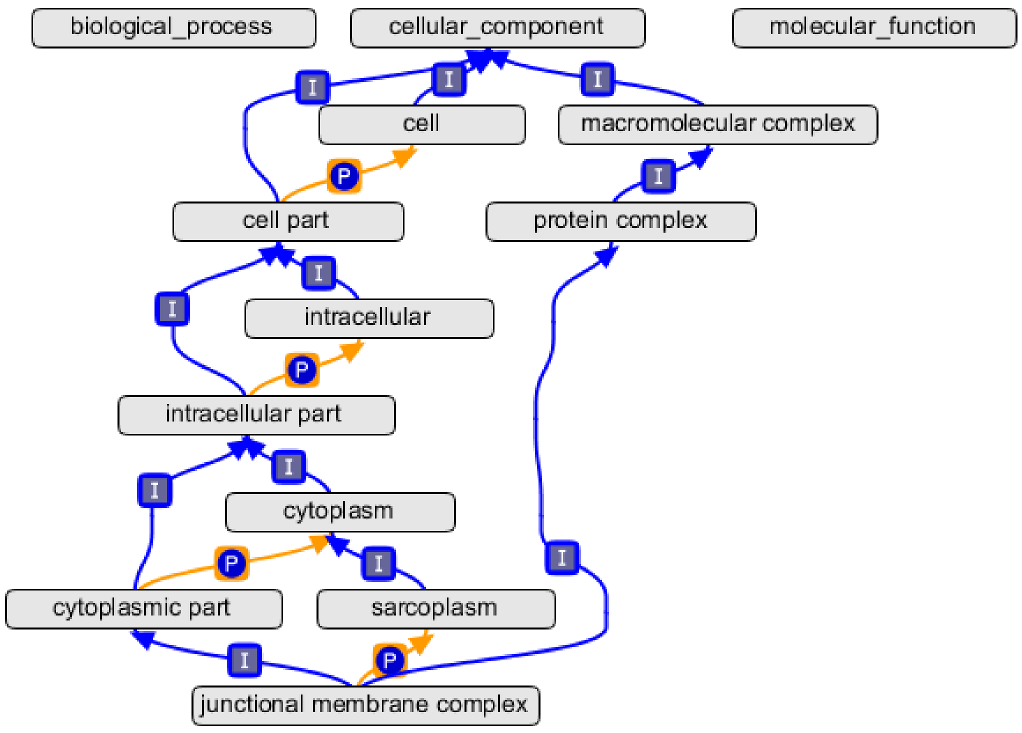


Figure 2.3: GO subgraph

ships. This means that following the `is_a` tag, any term can be traced back to one of the three main terms representing the ontologies of GO. This means the structure of the GO can be viewed as a directed acyclic graph with three distinct roots. A subsection of this graph leading from the GO:0030314 term to one of the roots is shown in figure 2.3.

Chapter 3

Design

This chapter provides a look into existing approaches to the issue of visualization of SBML documents and a clear specification of requirements expected from the system is given. Finally technologies are surveyed and the ones most suitable for the system are chosen.

3.1 Overview of existing software for SBML visualization

A lot of software working with SBML documents presents the users with a more or less detailed visualization of the network represented by the model yet generally the main concern of these programs are simulations performed on the network and not the visualization itself. Of all the programs listed in the SBML software matrix those freely available and having a visualization output for SBML models (and of course those that did not crash after loading the document[1]) were surveyed. Some interesting results are briefly presented next.

Biological networks[2] is a powerful tool enabling integrative analysis of interaction networks, metabolic and signaling pathways, proteomic data, genomic sequences, and multiple ontologies. The environment (figure A.1) seems promising but when working specifically with SBML models most of the functionality seems restricted. The compartment is shown and inside it all nodes of the network are drawn in a circle layout which becomes increasingly confusing with the size of the graph while the layout changer does not seem to affect them. The nodes are labeled by their SBML identifier and lack any other description.

As a part of **Systems Biology Workbench**[3] the Network Layout (figure A.2) is a simple, fast, efficient single purpose tool. It supports a single

1. All documents in this survey were downloaded from the curated biomodels repository at `http://www.ebi.ac.uk/biomodels-main/publmodels`
2. available at: `http://www.biologicalnetworks.org/`
3. `http://www.sys-bio.org/sbwWiki/doku.php?id=sysbio:downloads`

layout called "Auto Layout" which seems to be efficient even for larger networks. The interface enables repositioning of single nodes. Nevertheless the program does not offer any extra features to make the graph more comprehensible.

CellDesigner[4] a structured diagram editor (figure A.3) that can browse and modify existing SBML models. The program has a rich functionality but does not provide a clear view of the network that would help understand its functionality.

The **VANTED**[5] system, shown in figure A.4 exceeds the previous programs by its rich set of layouts that can be used to visualize a network. Although the layouts work independently and it is not possible to select a subgraph in the network which would be found sufficiently distributed and apply the layout on the rest of the network.

Programs with similar capabilities as the ones mentioned are E-cell[6]; BioSpice[7] and others.

3.2 Observations

It is very common for the SBML models to consist of only one loosely labeled compartment which does not add information value that can be further used for graphical representation. Even when the compartments are used to specify the location of nodes in the cell, other then their names, there is no information to reference their mutual alignment. For example we can have a compartment labeled cytoplasm and a compartment labeled nucleus. While it is evident for the human observer that these compartments should be positioned within each other a program often overlays these compartments inaccurately.

The SBML source primarily encoding biochemical reactions does not provide much information on the individual species other then their initial value and unit. For this reason all the programs mentioned distinguish only two types of nodes - species and reactions. Displaying reactions as nodes does carry informational value since there are many kinds of reactions. For example two species may act as reactants and produce a single product in a synthesis reaction, or the other way around a specie may be decomposed into more products. In the context of a cell, even reactions with no products or no reactants make sense as loose amino acids from the cy-

4. `http://www.celldesigner.org/`
5. `http://vanted.ipk-gatersleben.de/index.php?file=doc0.html`
6. `http://dev.e-cell.org/`
7. `http://biospice.sourceforge.net/`

toplasm may be picked up to form a protein. This process being reversible forming the other mentioned reaction.

3.3 Requirements

The visualization program developed here should exceed those mentioned previously. One of the big issues being the lack of information on the species of the SBML document it is expected that the program will harvest as much information from the annotation of species as possible. One of the main sources of information being Gene Ontology, the program should find the GO id of a term whenever it is possible. Other sources of useful information being databases like UniProt[8] (often seen in the annotation of species). This database also contains information on the location of proteins in a cell and more importantly each entry usually carries a cross reference to Gene Ontology. Once a specie is sufficiently identified it is then required to place it correctly into the structure of a cell and use a proper image representation. This representation should be specifiable by the user (given as a set of SVG images).

The requirement of semi-automation means that user interaction should not be required but can be expected. As the program generates a certain layout the user should be able work further with this visualization. For example he should be able to specify the nodes of which location is satisfactory and relocate the rest (either manually or by choosing a different layout algorithm).

The layout should adapt to the different parts of the network representing certain cell components and structure the nodes of the graph proportionally. It is also required that the visualization is comprehensive and aesthetically appealing which means the spacing of the nodes in the graph should be uniform (with respect to the subcomponents) and the crossing of edges should be minimal.

Finally the program should be able to convert and store the resulting visualization in the SVG format.

3.3.1 Requirements Summary

- Semi-automated visualization
- Adaptive distribution of cell components

8. UniProt is a freely accessible resource of protein sequence and functional information found at: `http://www.uniprot.org/`

- Comprehensive graph layout representing the biochemical network
- Gathering and work with Gene Ontology information
- Selective image representation
- SVG output

3.4 Survey of technologies to be used

The criteria of this analysis are functionality, good documentation and an open-source license.

3.4.1 Parsing SBML documents

Being an XML derivate, parsing SBML documents is a straight forward task. The simplest way to do this is to make use of the libSBML[9] library. This library if free, open source, and is meant primarily for reading, writing and validating SBML files.

Importantly libSBML has application programming interfaces for most major programming languages (including Python and Java)

3.4.2 Working with graphs

Graphviz[10] is an open source graph visualization software with many applications in networking, bioinformatics, software engineering and others. It works with simple text language (Dot language) descriptions of graphs and produces output in useful formats, such as images, SVG for web pages, PDF and GXL. Graphviz supports a whole variety of different layouts for directed and undirected graphs. Many programming languages have an interface to Graphviz (PyDot[11] for Python).

The drawback and the main reason why Graphviz was not chosen for the project was the dysfunctional feature of setting the coordinates of nodes positions and using the pin-function to lock this position for further layouting.

After experimenting with other main graphing libraries like NetworkX[12] and igraph[13] (python extension) which were ruled out because of com-

9. http://sbml.org/Software/libSBML
10. http://www.graphviz.org/
11. http://code.google.com/p/pydot/
12. http://networkx.lanl.gov/
13. http://igraph.sourceforge.net/

plicating work with insufficient documentation, the slightly different approach of the open source JUNG[14] library written in Java proved to be sufficient to solve the stated requirements.

JUNG stands for the Java Universal Network/Graph framework. It is a software library that provides a common and extendible language for modeling, analysis, and visualization of data that can be represented as a graph or network.

A graph in JUNG is represented by the interface Graph<V,E>where V represents the type of the vertices and E the type of edges defined by the user. For the case of a graph representing an SBML model it is sufficient to use the id of a specie or reaction which can be mapped to a structure representing selected attributes of the node.

To set coordinates to the nodes of a graph JUNG uses the *Layout* interface which is a utilization of the *Transformer<V,Point2D>* interface. The transformer idiom is very common in JUNG. It is used in various layout and graph algorithms to extract information from edges and nodes such as edge weights and node scores. In this case the Layout transformer translates nodes to their position coordinates. Additionally the Layout interface provides mechanisms to initialize the location of all vertices in a graph, set the location of a particular vertex, lock or unlock the position of a vertex, and more. JUNG contains many useful implementations of the Layout interface such as spring algorithm layout, Kamada-Kawai algorithm, Fruchterman-Reingold force-directed algorithm, a simple circle layout algorithm, and others.

Finally to show a graph one of the implementations of the JUNG's *VisualizationServer* interface are used. Serving as a 'canvas' for drawing the graph, these implementations also inherit from Swing's JPanel class and can be placed anywhere in the Swing GUI hierarchy.

Because of their frequent use in bioinformatics, Python and Java are the main candidates for the programming language to be used in the development process, choosing JUNG answers this open question.

3.4.3 Gathering Data

Once the input file is parsed and the process of building the graphical representation has started it is important make use of any data available that could guide the visualization process.

To access the data from the Gene Ontology it is efficient to download

14. `http://jung.sourceforge.net/doc/index.html`

the current version of the database in the text based OBO[15] file format. This file can be parsed and the ontology represented as a directed acyclic graph. Once the graph representation is built, location information can be accessed by traversing the graph backwards (from the node representing a specie to the root node) and searching for representations of subcellular locations and components. For example the all the successors of the term GO:0044428 "nuclear part" can be expected to be located in the nucleus of a cell. As more than one paths from a component to the root can be expected, the most specific information should be chosen (on the path to the node closest to the component).

Other information such as UnipProt database entries can be accessed through an online retrieval service.

3.4.4 Working with Images

Working with SVG images occurs on two places. First it is the user specified SVG library for node representation. Allowing the user to specify image representations means that he should be able to specify which components to display and what representation to use. It seems suitable that the images to be displayed are identified by the Gene Ontology Id. This way the same process of searching in the GO graph can be applied as is used to set the component location. It also seems sensible to distinguish the different kinds of reactions with different representations.

The second place where imaging is used is the final output of the graph to an SVG image.

Since JUNG does not support SVG images another library is required to handle these demands. Batik[16] - a Java based toolkit for working with SVG formats satisfies all these requirements. It enables conversions to other image formats (for node representing image input) and can be used to build an SVG document from the VisualizationServer information.

3.5 Creating an adaptive compartmental layouting algorithm

JUNG does have a class called AggregateLayout but upon examination it can be seen that this class does not sufficiently solve the problem of a compartmental layout required by the nature of the cell structure. A solution to this problem is proposed next.

15. http://www.geneontology.org/GO.format.obo-1_2.shtml
16. http://xmlgraphics.apache.org/batik/

Firstly all the necessary compartments must be identified. This can be done by the information gained from Gene Ontology where each specie (future node of the graph visualization) carries its compartmental association. This way it is possible to calculate the space each node can theoretically occupy.

$$\frac{Space}{All\_nodes} = Unitary\_value$$

The size of compartments then can be derived by the following formula. For each compartment C:

$$Size_C = Unitary\_value * \sum Nodes_C$$

Knowing the count and size of compartments, the next step is to position them accordingly. At this point the standard structure of a cell should be considered as a guideline for compartmental layout. Once the compartments are in place the nodes of the graph should be spread out inside their own compartment but simultaneously cross-compartmental interactions must be respected. Taking advantage of an iterative layouting algorithm this process can be decomposed into repetitive process consisting of the following steps:

- Inner-compartment layouting
- Inter-compartment layouting
- Recalibrating step making sure the compartmental boundaries are respected

The algorithms terminating point can be determined by the overall amount of repositioning[17] carried out in one repetition. Such a layouting algorithm should sufficiently solve the compartmental layouting problem.

3.6 Design summary

- Utilize the libSBML library to parse the input SBML document.
- Gather available information from Gene Ontology and the Uniprot Databse.
- Use JUNG library for graph representation and visualization. Represent both species and reactions from a model as nodes.

17. For example the distance to which the nodes are moved.

- Make use of JUNG's layouting algorithms and modify them to work on a compartmental model.
- Exploit the Batik library for work with SVG images.

Chapter 4

Implementation

The previous chapter states the goals for the program to be written and gives some insight into how this should be done.

Now the actual results of the implementation, source code analysis, and the working program are presented.

4.1 Program run and source code analysis

The most important class of the program is `SBMLGraphVis`. It contains the `main()` function, builds the graphical user interface, holds the graph representation of the network, and contains the layout algorithm.

At initialization an empty graph representation is created and JUNG transformers are added to the visualization so that the nodes of the future graph can contain custom images, labels and tooltip information. The example source code enabling all this is shown in listing4.1

Following the initialization the main function instantiates a `GOGraph` class which serves as the representation of Gene Ontology for the program. The GOGraph class parses the OBO document utilizing an OBOParser[1]. Even though the inner representation of Gene Ontology is not actually a graph it possesses the functionality proposed in the design section and is treated so.

After the Gene Ontology file is loaded the user is encouraged to load the folder (using a standard Java file chooser dialog) containing the set of images he chose to represent the graph components. The images should be labeled by a GO id (without the "GO:" prefix, using only the number). Special names are reserved for the "default" image to be used when no other image applies and three names for different reaction types. These are "react01" for the simplest reaction with no reactant and no product; "react11"

1. Tho OBO parser is part of the open source, Java implemented OBO-Edit - a gene ontology editor program available at: `http://sourceforge.net/projects/geneontology/`

Listing 4.1: Creation of a graph representation, source code example

```java
Graph<String , Number> sGraph;
Layout<String , Number> layout;

sGraph = new DirectedSparseGraph<String ,Number>();
layout = new SpringLayout<String ,Number>(sGraph);
layout.setSize(new Dimension(1200,500));

VisualizationViewer<String ,Number> vv;
vv = new VisualizationViewer<String ,Number>
          (layout, new Dimension(1300,600));

Transformer<String ,Paint> vpf =
          new PickableVertexPaintTransformer<String>
          (vv.getPickedVertexState(), Color.white,
          Color.yellow);
vv.getRenderContext().setVertexFillPaintTransformer(vpf);
vv.setBackground(Color.white);
vv.setVertexToolTipTransformer(new
          VoltageTips<String, Object>());
vv.getRenderContext().setVertexLabelTransformer(new
          ToStringLabeller());
vv.getRenderer().getVertexLabelRenderer().setPosition(
          Position.S);
```

for simple "one to one" reactions; and "reactin" for all other types of reactions. (For example in the default set of images used during development the reactions with no reactants or products are displayed as a cloud icon which seems appropriate for representing the nature of these kinds of reactions).

Once the image library is loaded the user selects an SBML file he wishes to visualize. The loading of this file launches a whole set of processes. Firstly the sbml document is parsed by libSBML. The program creates a JUNG graph representation of the network as each specie is read. In this graph the vertex is represented by a string. More precisely it is the id of the specie (or reaction). This string is mapped to a class representing the node called sNode. The sNode class contains all the information about the node and methods to retrieve these information. According to the information found in the annotation it either calls the GOGraph parse method with the GO term as the parameter or the online Uniprot data retrieval service with the found Uniprot Id is called.

The GOGgraph parse method which works as proposed in the previous chapter . As shown in listing4.2 the method first tries to find an image representation for the term and identify its location(this part of source code is

Listing 4.2: Gathering Gene Ontology information, source code example

```
public void parseGOTerm(SBMLGraphVis vis,sNode node){
    String goId = node.getGoId();
    LinkedList<LinkedObject> parentQueue =
        new LinkedList<LinkedObject>();
    if(goId==null)
        return;
    IdentifiedObject obj = db.getObject(goId);
    Collection<Link>lcol=db.getParents((LinkedObject)obj);
        for(Iterator<Link>i=lcol.iterator();i.hasNext();){
            Link l = i.next();
            l.getParent();
            parentQueue.add(l.getParent());
        }
        while((!node.has_image()||!node.isTypeLocked())&&
                                   !parentQueue.isEmpty()){
            String imagePath = vis.getImage(obj.getID());
            System.out.println(imagePath);
            //set Image
            if(!imagePath.equalsIgnoreCase("none")&&
                                   !node.has_image())
                node.setImagePath(vis,imagePath);
            obj = parentQueue.remove();
            lcol= db.getParents((LinkedObject)obj);
        ...}
    ...}
}
```

omitted for brevity). If this is not successful the search continues one step higher in the Gene Ontology Graph (the parents of the current term). Once the image (or location) is set for the node it is locked so that in case the search continues, another setting of a less specific representation (or location) is prevented.

The entry received from Uniprot contains information in the form of comments. If they exist the comment of type `SUBCELLULAR_LOCATION` and a GO id are retrieved. The location comment is used to set (and lock) the compartmental location of the node being parsed and the GO id is used for further GOGraph parsing and image association.

Having acquired all the information the program must determine which components of the cell structure are to be shown and what should their dimensions be. There are six compartments representing different subcellular components based on the most commonly used subcellular parts in the SBML documents. The compartments are illustrated in figure4.1

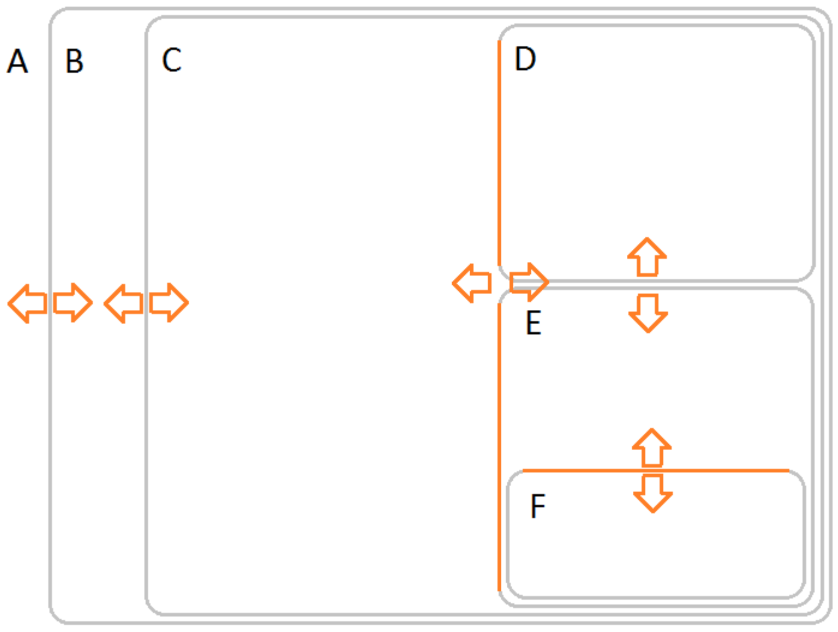


Figure 4.1: Compartment Diagram[2]

2. The compartments represent A: The outside of a cell for extracellular components;B: Cytoplasmic membrane;C: Cytoplasm; D: Mitochondrion; E: Nucleus; F: Nucleolus

The arrows in the diagram represent how the layout adapts to the model. The size of each compartment is calculated by a simple formula which divides the whole space available by all the nodes representing species which gives a unit number which is then multiplied by the nodes of each compartment giving the resulting size.

The borders of the compartments outlined in orange color show where nodes representing membrane parts are to be placed.

4.1.1 Layouting algorithm

Once the compartments are set and each node has its compartmental adherence the program starts the graph layouting process. By default the JUNG Spring algorithm is used where the nodes are repulsed by each other by a force depending on their distance and simultaneously the edges of the graph work as an attractive force pushing connected nodes near each other. As the algorithm works on an iterative basis it was possible to modify it as proposed in the design section. In each repetition the algorithm makes one iteration spreading the nodes of the graph and is then recalibrated by the checkCompartmentBounds() function. The abbreviated source code of the

Listing 4.3: The main cycle of the layouting algorithm, source code example

```
layout.initialize();
for(int iter = 0;iter<=1000;iter++){
    ((IterativeContext)layout).step();
    checkCompartmentBounds(layout);
}
Collection<String> vertSet=sGraph.getVertices();
for(Iterator<String> i=vertSet.iterator();i.hasNext();){
    String vert = i.next();
    sNode node = nodeMap.get(vert);
    if(node.getType()!=sNode.Type.REACTION){
        locMap.put(vert, new Point2D.Double(((
            AbstractLayout)layout).getX(vert),
            ((AbstractLayout)layout).getY(vert)));
        layout.lock(vert, true);
    }
}
```

algorithm is shown in listing4.3

Afterwards when the locations are set they are added to a map of locations and locked. Further on this lock can be manipulated with by the user. The user may wish to redo the generated layout. At this point there are two choices. The layout can be redone by applying another of JUNG's layouting algorithms (also modified to respect the compartmental layout) and/or by repositioning the nodes individually.

4.2 Graphical User Interface

Figure4.2 shows the program displaying a model. A set of default images was used to represent the species and reactions. The model utilizes two cell compartments, cytoplasm and the nucleus. The green tick represents the lock on the nodes position it can be added or removed simply by clicking on the node. The nodes can be moved around the graph with the mouse. Nodes of Species are labeled by their name but more information on both specie and reaction nodes can be gained by hovering the mouse tip over a node.

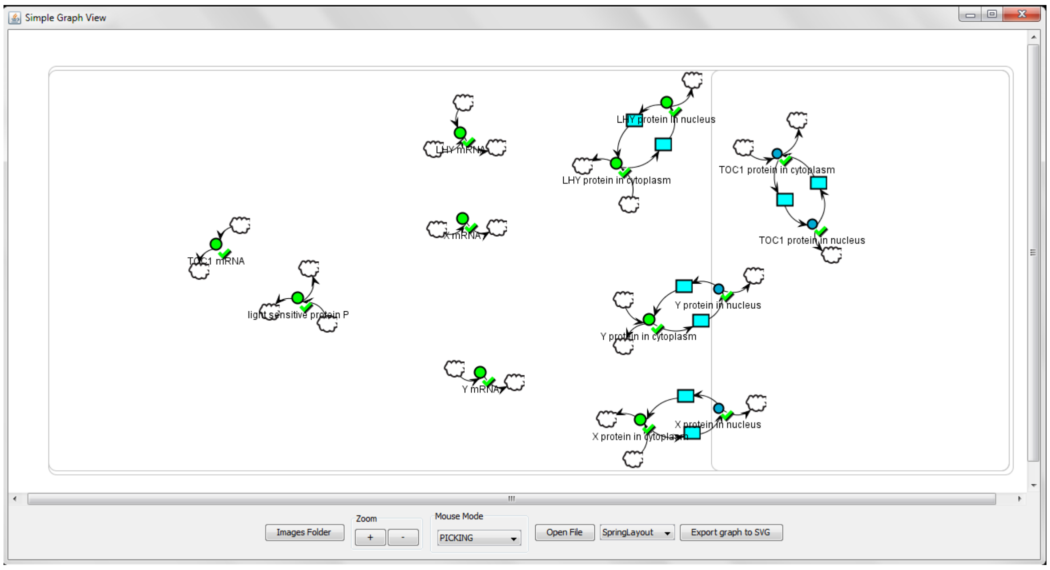


Figure 4.2: Program example run

The "Image Folder" button opens a Java open file dialog window which enables specifying the folder containing the set of images for node representation. The zoom buttons execute a straight forward zoom in and zoom out function. There are two ″mouse modes″ in which either the whole graph is edited (moved, rotated and morphed) or the separate (single or multiple) nodes are selected, moved and locked. The ″open file″ button springs another open file dialog where the user specifies the SBML model. The combo box contains a small list of different JUNG implementations of layouts. By clicking on a layout the graph is redrawn leaving the locked nodes in place and respecting the compartment boundaries (Except for the reaction nodes, they can be locked but they are not assigned to any compartment for practical reasons). Finally the ″Export″ button springs a save file dialog where a name and location can be specified for the output SVG image of the graph to be saved.

Chapter 5

Case study

The following section is a short discussion on a particular diagram generated by the program which was created as discussed in the previous chapter. The diagram was generated from a moderately complex network[1].

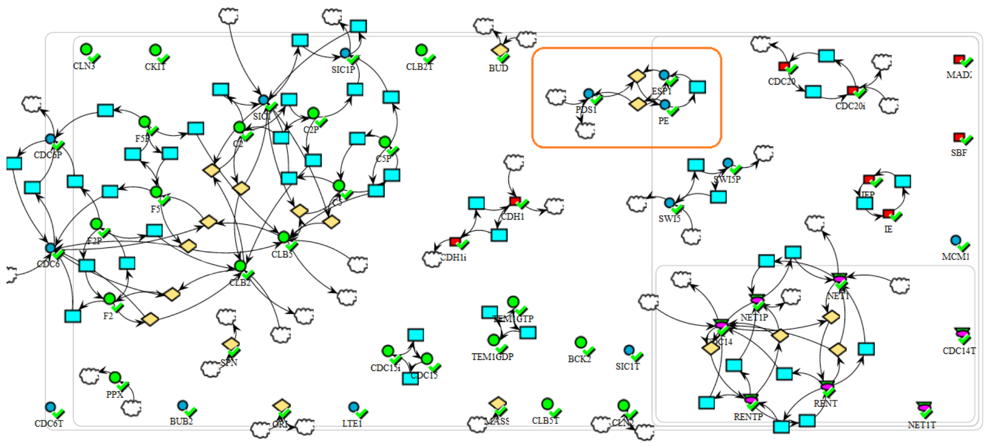


Figure 5.1: An Example SBML model visualization generated by the developed program.

In the diagram we see that 4 compartments were utilized and that all the nodes representing species respect the boundaries of the compartment they were assigned to. A nice example is the subgraph outlined in the orange rectangle[2] where the model has adapted to a subgraph of which nodes belong to two different compartments. The compartments size is proportional to the nodes held inside them. The overall spreading of nodes is more or less

1. The model "SBML model of Cell cycle control mechanism" was used. It was downloaded from BioModels Database. The file name is BIOMD0000000056.xml.
2. The orange rectangle is not part of the original output. It serves the purpose of outlining an interesting part of the graph.

uniform (makes use of the space available) and as the result of the spring algorithm unattached subcomponents of the graph were pushed away from each other which results in a more readable visualization.

For comparison the following image is diagram created by the CellDesigner program. In this visualization the same model was used as the one in the example case.

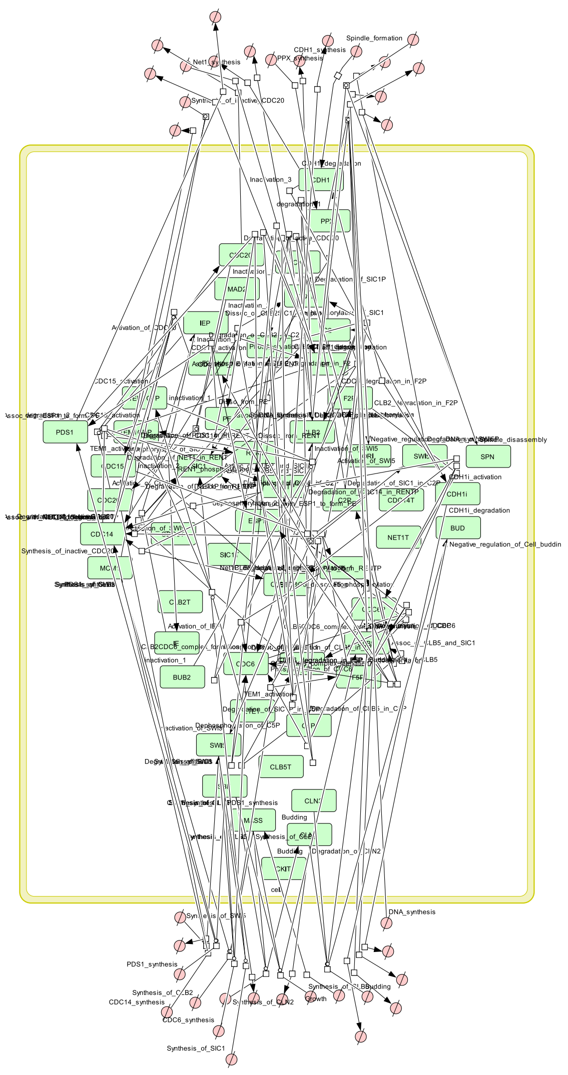


Figure 5.2: The output of CellDesigner.

Chapter 6

Conclusion

The widely used SBML format consists mainly of machine generated information. The main ambition of this thesis was to bring out this information in an more human comprehensible intuitive manner of diagram representation.

First the reader was familiarized with the main concepts this thesis works with, and the problematics it is trying to solve. Next a list of inspirational solutions was given and after pointing out their weaknesses a proposal on how to solve them was shown. This proposal surveys the technologies that could be exploited to help solve some of the problems and presents the main ideas behind the solutions to the rest.

Afterwards the program implementation based on the preceding proposal is illustrated. The run of the program is explained while giving comments on some pieces of the source code and explaining the main ideas of the solutions in the context of the functional program. The GUI is then presented to the user and a short manual like description is given.

Finally an example output presenting the capabilities of the program is shown, explained and compared with the result of another similar program.

Some ideas for the future work in this area are a three dimensional visualization for a model or an animation visualization showing an interactive simulation on such a model.

Bibliography

[1] Autumn A. Cuellar, Catherine M. Lloyd, Poul F. Nielsen, David P. Bullivant, David P. Nickerson, and Peter J. Hunter. An overview of cellml 1.1, a biological model description language. *SIMULATION*, 79(12):740–747, 2003.

[2] M. Hucka, A. Finney, B.J. Bornstein, S.M. Keating, B.E. Shapiro, J. Matthews, B.L. Kovitz, M.J. Schilstra, A. Funahashi, J.C. Doyle, and H. Kitano. Evolving a lingua franca and associated software infrastructure for computational systems biology: the systems biology markup language (sbml) project. *Systems Biology*, 1(1):41–53, 2004.

[3] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, , the rest of the SBML Forum:, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novďż˝re, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[4] Michael Hucka, Andrew Finney, Stefan Hoops, Sarah Keating, and Nicolas Le Novere. Systems biology markup language (sbml) level 2: Structures and facilities for model definitions. *Nature Preceings*, 2007.

[5] ION I. MORARU, JAMES C. SCHAFF, BORIS M. SLEPCHENKO, and LESLIE M. LOEW. The virtual cell. *Annals of the New York Academy of Sciences*, 971(1):595–596, 2002.

[6] Bruce E. Shapiro, Andre Levchenko, Elliot M. Meyerowitz, Barbara J. Wold, and Eric D. Mjolsness. Cellerator: extending a computer algebra system to include biochemical arrows for signal transduction simulations. *Bioinformatics*, 19(5):677–678, 2003.

[7] M Tomita, K Hashimoto, K Takahashi, T S Shimizu, Y Matsuzaki, F Miyoshi, K Saito, S Tanida, K Yugi, J C Venter, and C A Hutchison. E-cell: software environment for whole-cell simulation. *Bioinformatics*, 15(1):72–84, 1999.

[8] Marketa Zvelebil and Jeremy O. Baum. *Understanding Bioinformatics*. Garland Science, Taylor & Francis Group, New York,N.Y., 2008.

Appendix A

SBML Visualizations by different programs

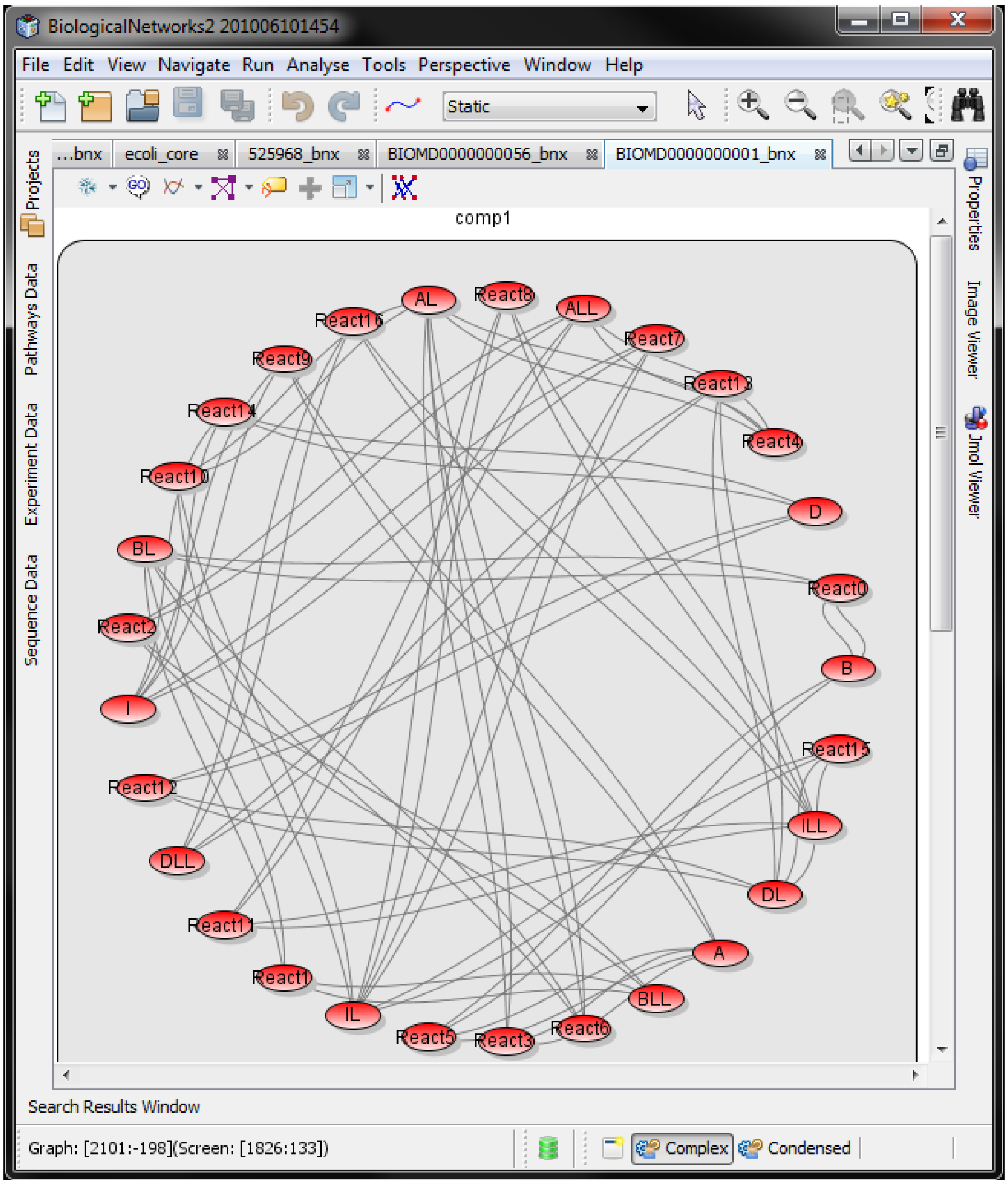


Figure A.1: Visualization of an SBML model by Biological networks from `http://www.biologicalnetworks.org`

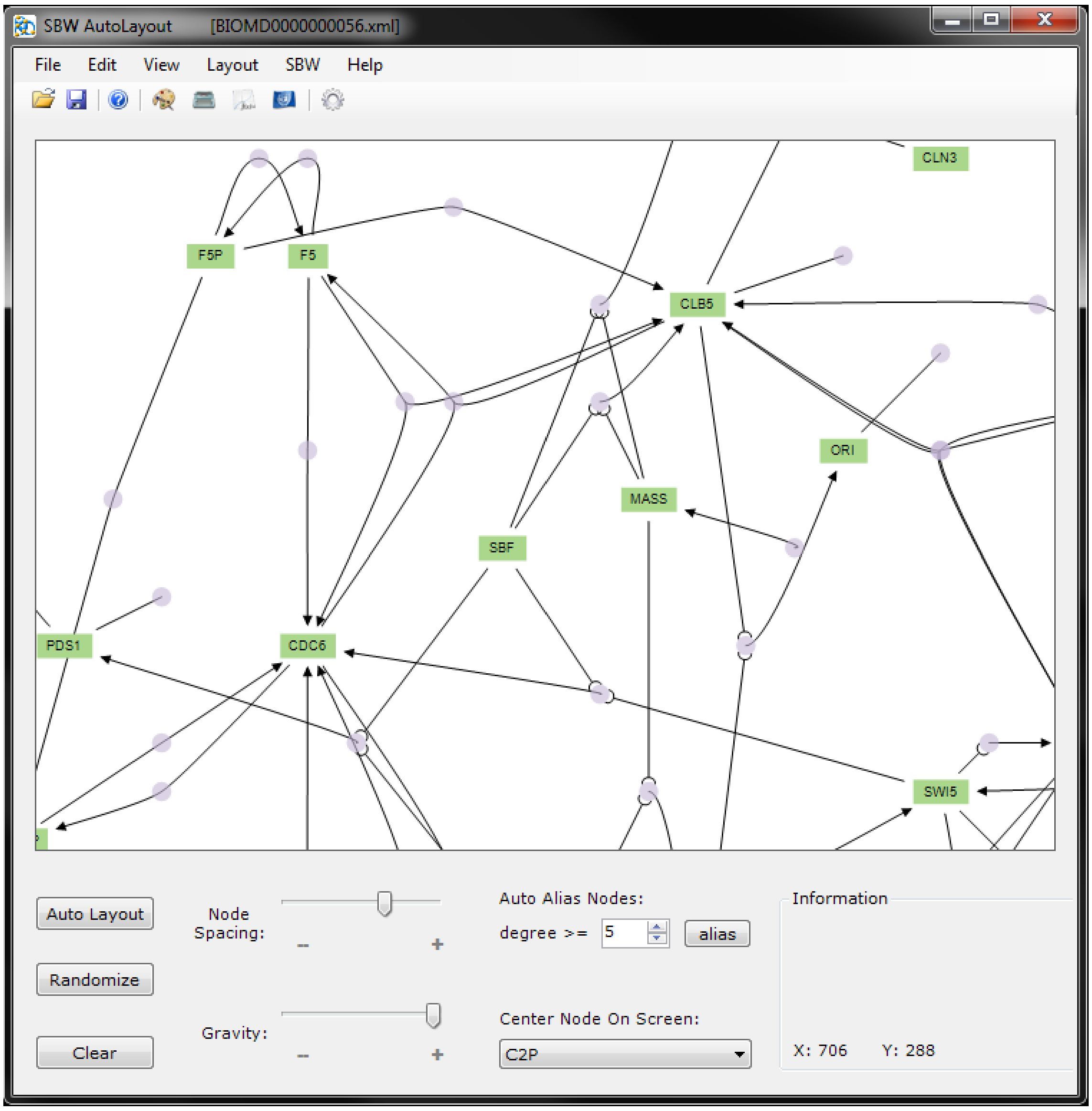
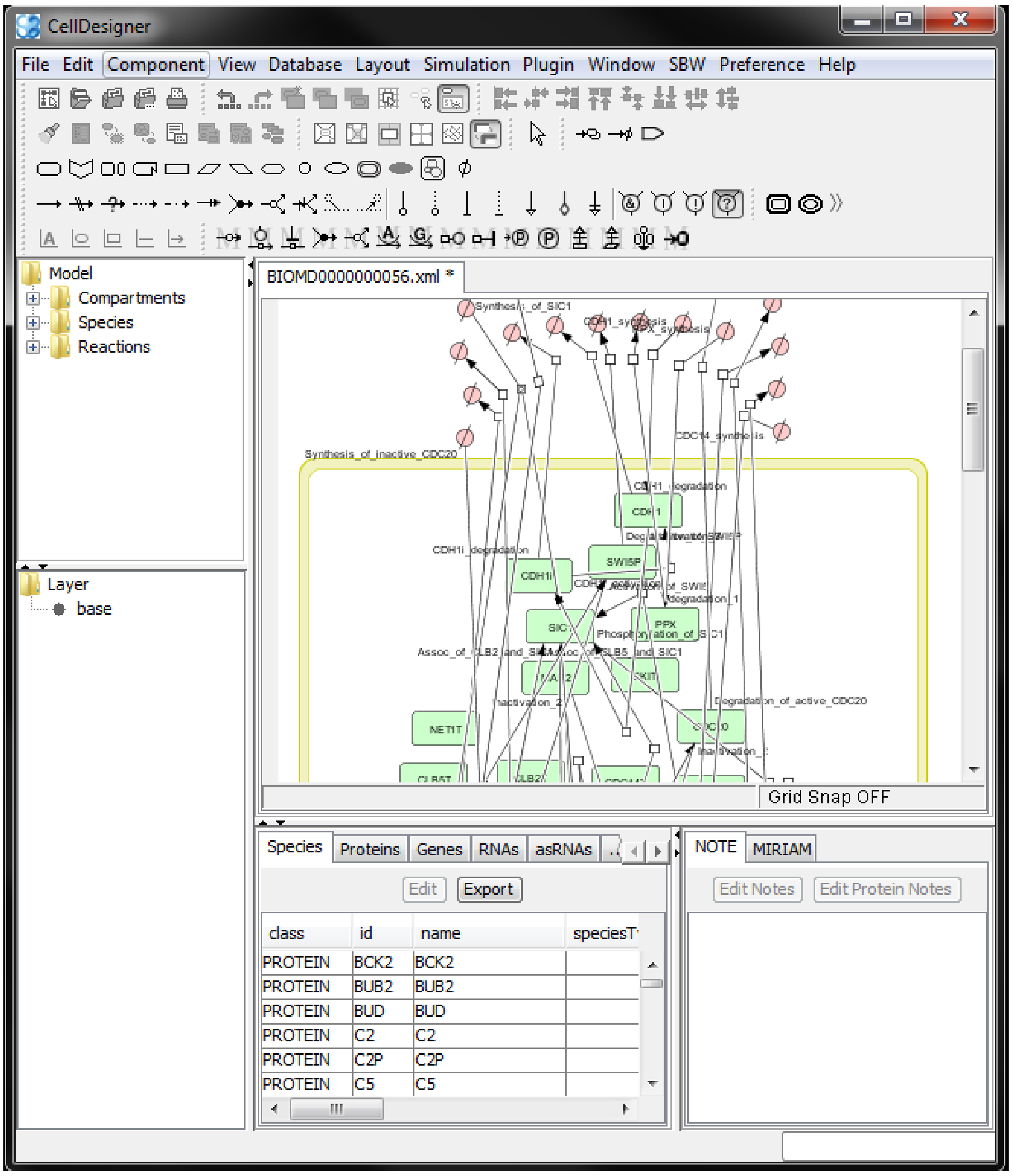


Figure A.2: Network Layout a part of Systems Biology Workbench



class	id	name	speciesT
PROTEIN	BCK2	BCK2	
PROTEIN	BUB2	BUB2	
PROTEIN	BUD	BUD	
PROTEIN	C2	C2	
PROTEIN	C2P	C2P	
PROTEIN	C5	C5	

Figure A.3: CellDesigner

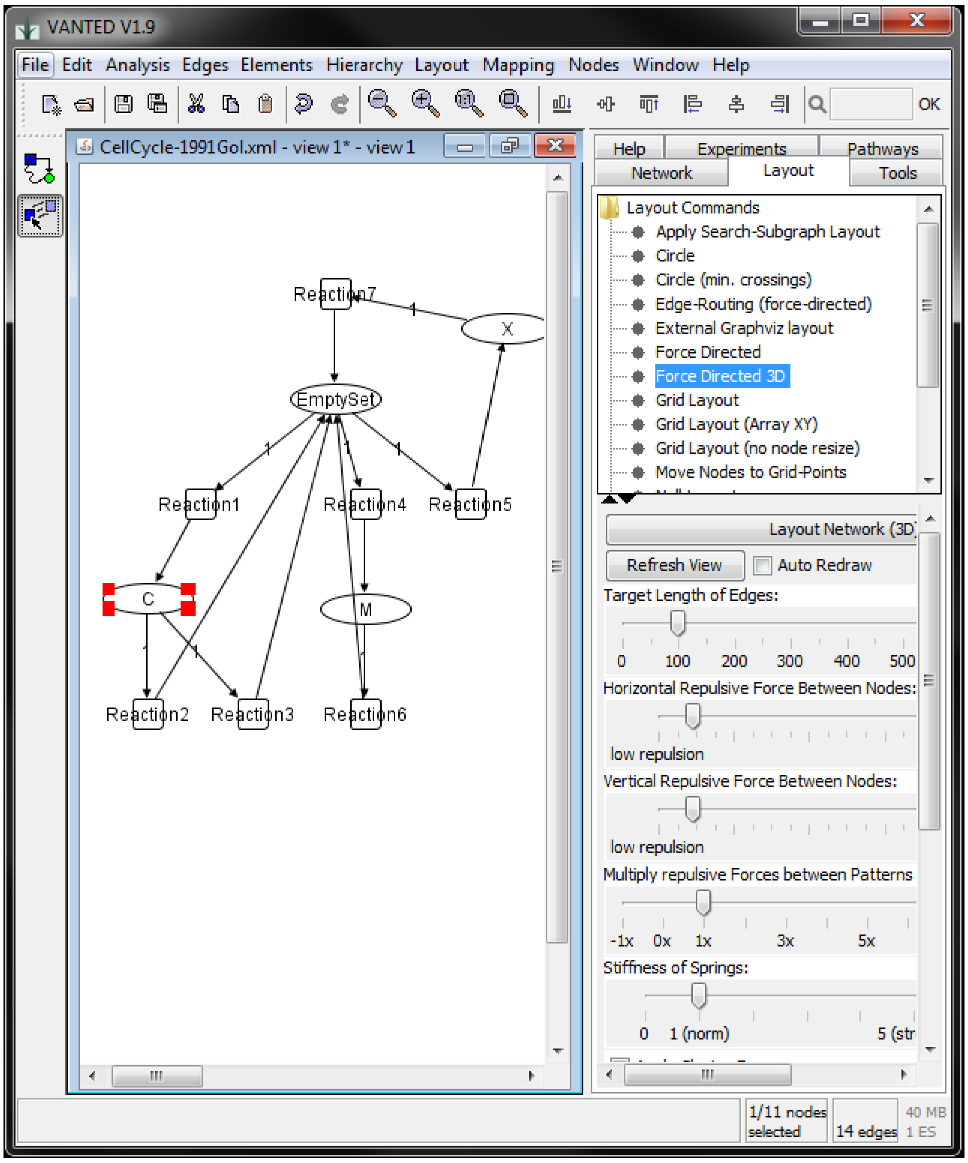


Figure A.4: VANTED - Visualization and Analysis of NeTworks containing Experimental Data

Appendix B

CD Contents

The attached CD contains the following:

- GraphVis.zip - The source code of the created program.
- diploma_thesis.pdf - The text of this thesis.