Institute of Systems Biology

BioUML workbench 0.8.6

[user guide]

Contents

1 Introduction

'We now have unprecedented ability to collect data about nature but there is now a crisis developing in biology, in that completely unstructured information does not enhance understanding. We need a framework to put all of this knowledge and data into - that is going to be the problem in biology.

We've reached the stage where we can't talk to each other - we've all become highly specialized. We need a framework, a framework where people can come back to us and say, 'Yes, I understand.' Driving toward that framework is really the big challenge.'

Sydney Brenner

We believe that BioUML - Biological Universal Modeling Language - is a step in this direction. It is imagined as a language to write a "book of life". **We hope that BioUML will be a platform for building virtual cell and virtual human**.

From the user's perspective BioUML workbench is integrated environment that spans the comprehensive range of capabilities including access to databases with experimental data, tools for formalized description of biological systems structure and functioning, as well as tools for their visualization, simulation, parameters fitting and analyses (Figure 1.1).

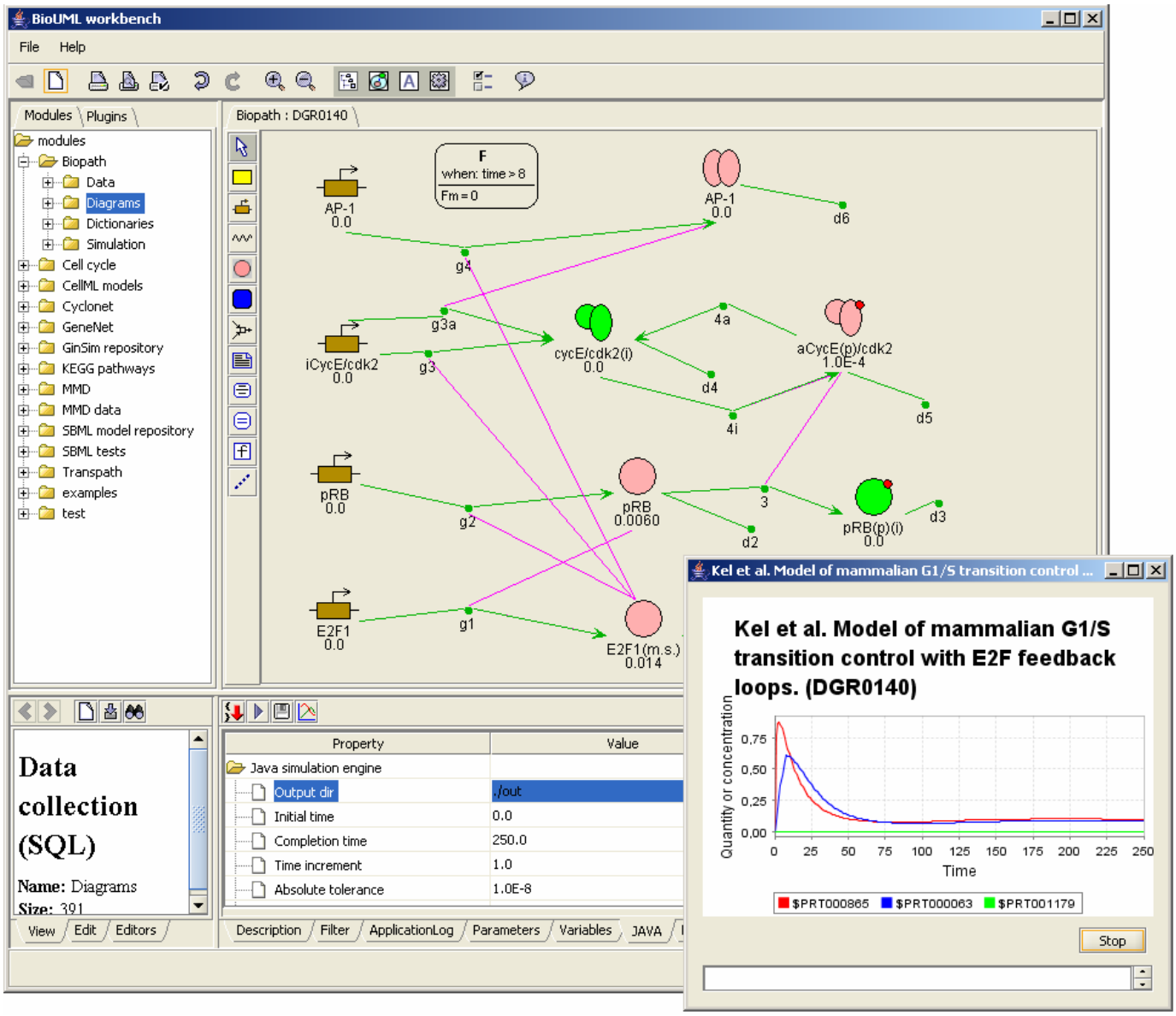


Figure 1.1. BioUML workbench - example of cell cycle model visualisation and simulation.

Below we will briefly consider main concepts and possibilities of BioUML workbench.

1.1 Visual modelling

Reconstruction of complex biological systems from a huge amount of experimental data requires a formal language that can be easily understood both by human and computer.

It is known that graphical depiction of complex system is the most suitable way of understanding of its structure by human. Graphical notation allows human to completely and formally specify model so computer programs can analyze the model and simulate its behavior (Lee, 2001).

This approach is widely used in engineering and computer science. Some examples are:

- MATLAB/Simulink (http://www.mathworks.com)
- AnyLogic (http://www.xjtek.com) - muli-method simulation software

- UML (http://www.omg.org/uml/) - the most known graphical language for computer science.

BioUML workbench adopts visual modeling approach for formal description and simulation of complex biological systems (Figure 1.2). Another distinctive feature of BioUML workbench is tight integration with databases on biological pathways, query engine allows user to find interacting components of the system and show results as an editable graph.

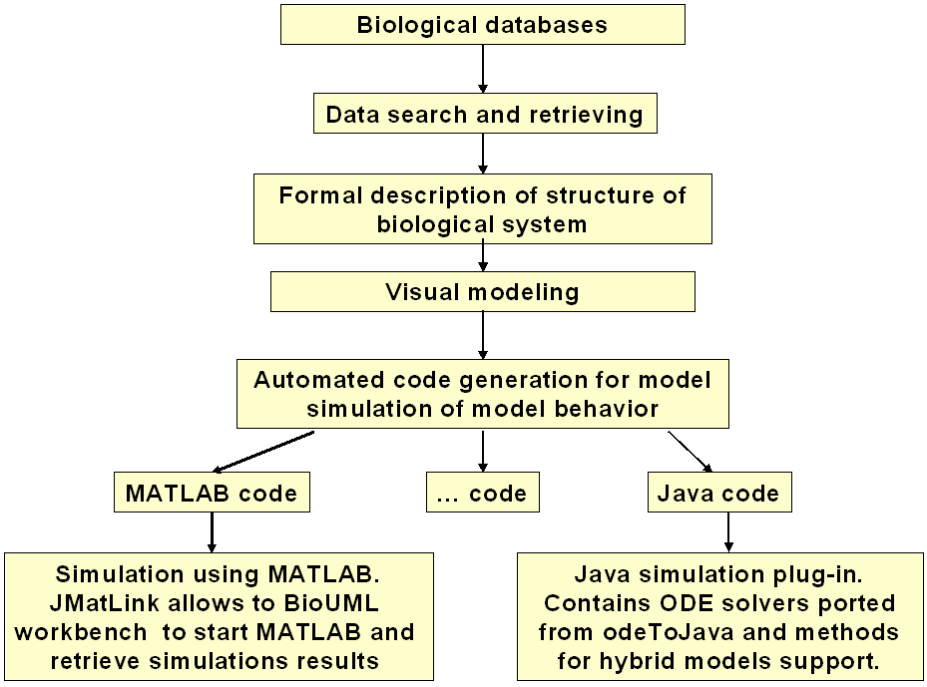


Figure 1.2. Data flow in BioUML workbench.

1.2 Meta model

The core of BioUML is a meta-model. It provides an abstract layer (compartmentalized attributed graph) for comprehensive formal description of wide range of biological and other complex systems. Content of databases on biological pathways, SBML (Hucka M. et al., 2003) and CellML (Lloyd C.M. et al., 2004) models, as well as biological pathways in BioPAX format can be expressed in terms of the meta model and used by BioUML workbench.

This formal description can be used both for visual depiction and editing of biological system structure and for automated code generation to simulate a model behavior.

Meta-model is problem domain neutral and splits the system description into 3 interconnected levels:

1. graph structure - the system structure is described as compartmentalized graph;
2. database level - each graph element can contain reference to some database object;
3. mathematical model - any graph element can be element of mathematical model.
   BioUML supports following mathematical elements: variable, formula, equation, event, state and transition.

Figure 1.3 demonstrates how this approach is applied for modeling system consisting from two consecutive chemical reactions. Here graph nodes representing chemical substances are considered as variables and corresponding graph edges contain right parts of corresponding differential equations. Using this information BioUML workbench can

generate MATLAB or Java code for model simulation.

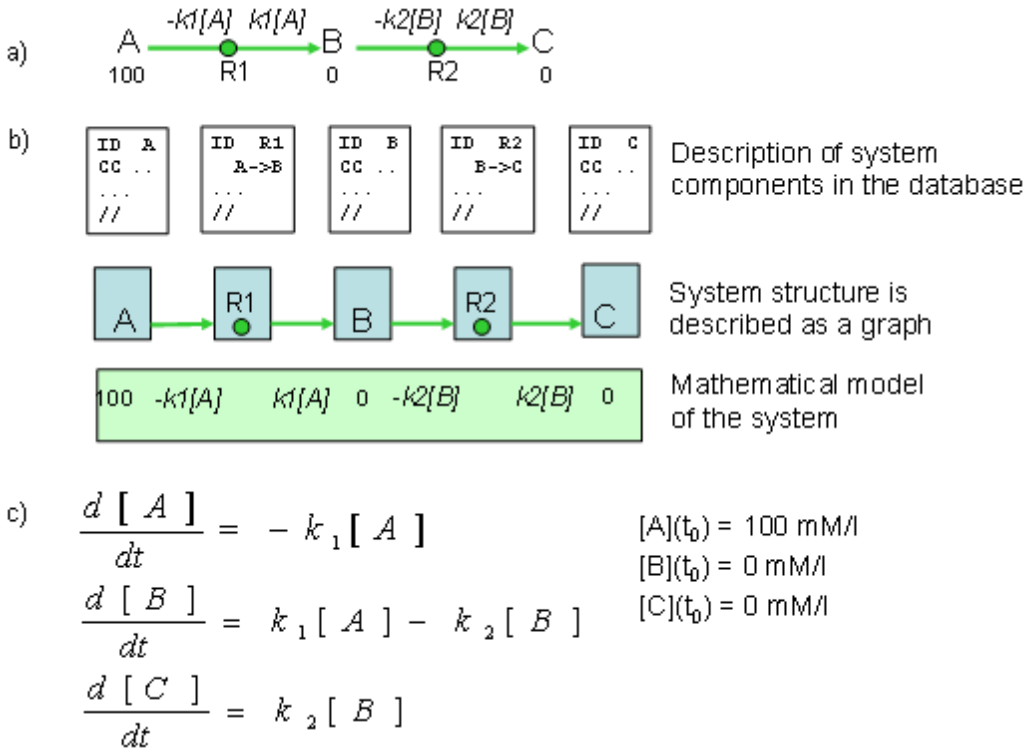


Figure 1.3. System from two consecutive chemical reactions (a), its formal description using three meta model levels (b), and corresponding mathematical model (c), that can be generated automatically for system simulations.

Special BioUML diagrams markup language (DML) is developed to store BioUML meta model instance in XML format. Diagram description is divided into two parts:

1) graph structure - it describes location of diagram elements and contains references to associated with them database objects;
2) executable model - stores mathematical model associated with graph.

Detailed description of DML format is available at http://www.biouml.org/dml.shtml

1.3 Diagram type

To take into account different diagram types and problem domain specificity we have introduced the diagram type concept.

Diagram type defines:

- types of biological components and their interactions that can be shown on the diagram;
- diagram view builder – it generates view (image) for each graph element taking into account the problem domain peculiarities. For example, biological pathway diagram view builder displays proteins as circles, genes as rectangles and substances as squares;
- semantic controller - provides semantic integrity of the diagram during its editing. It takes into account problem domain constraints, for example if some substance is removed on biological pathway diagram, all related reactions should be removed too.

Diagram type can be defined (created) by two ways:

1. programmatically - as Java class implementing special interface. There are 5 predefined diagram types that allows to describe complex biological systems on cellular level with different level of details and formality;
2. declaratively - as XML document. BioUML workbench provides Graphic Notation Editor that allows advanced user to create and edit diagram types. By this way diagram types for SBGN (Le Novère et al, 2009) and KEGG/ Pathways database were created (Figure 1.4).

BioUML workbench provides 3 main diagram types for modeling metabolic pathways, signal transduction pathways and gene networks:

1. Semantic network - this diagram type is used to describe semantic relationships between system components, system states, and related problem domain concepts. This diagram type is also convenient as overview.
2. Pathway - is used for formalized description of biological pathway structure (metabolic pathway or gene network).
3. Pathway simulation - is extension of pathway, where variables are associated with graph nodes and differential equations with graph edges. This allows to BioUML workbench automatically generate mathematical model of the system and simulate its dynamics.

4. **See also:**

- Chapter Diagram types
- Chapter Graphic Notation Editor

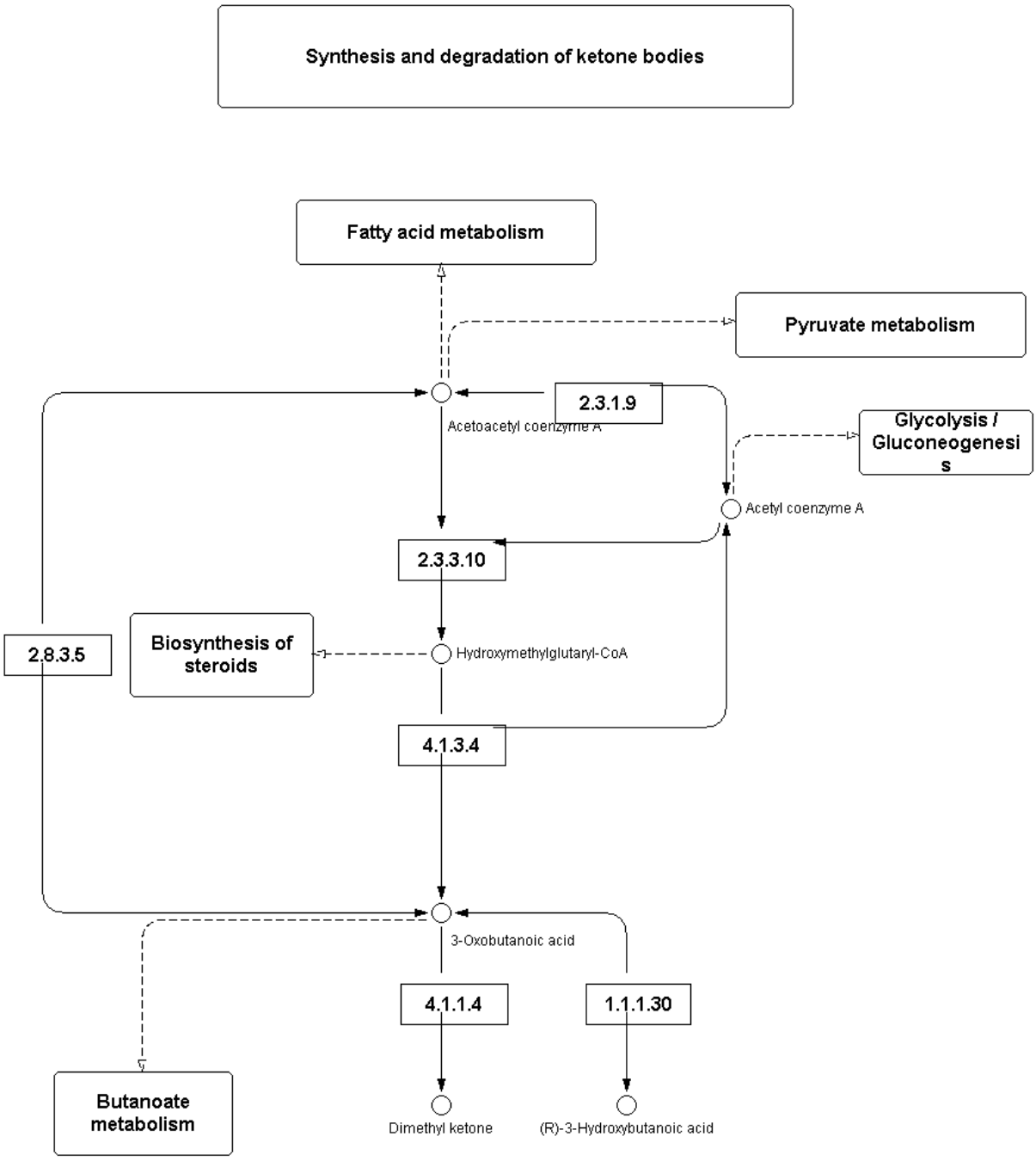


Figure 1.4. Example of diagram generated by BioUML workbench using KEGG graphic notation.

1.4 Simulation engine

BioUML workbench provides two alternative simulation engines:

1) Java simulation engine - it automatically generates and compiles Java code on the base of visual model (diagram) of a biological system. For simulation we have adopted odeToJava library (Patterson and Spiteri, 2003) that provides methods for numerical solutions both stiff and non-stiff systems of ODEs. For solving algebraic equations Newton solver is used.

2) MATLAB simulation engine - workbench automatically generates code for MATLAB and invokes MATLAB engine to simulate a model behaviour using JMatlink library (http://www.held-mueller.de/JMatLink/).

Main parts of simulation engine are: code generator, formulas processor, algebraic equations solver and results writer. BioUML provides powerful formula processor that parses text and MathML expressions, result is presented as syntax tree and used by formatters to generate corresponding Java or Matlab code (Figure 1.5).

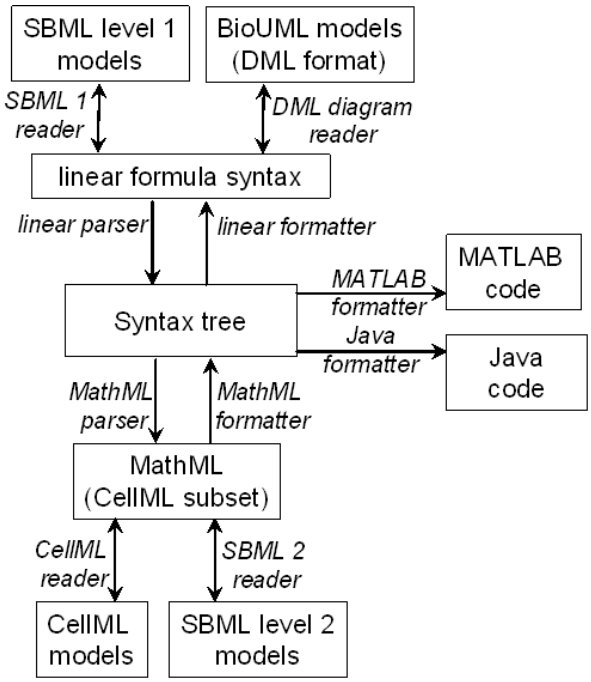


Figure 1.5. Parsing and conversions of mathematical expressions by simulation engine.

Both simulations engines pass SBML semantic test suite that provides a set of valid SBML models with a simulated time course data (Finney, et al., 2004). Test details are available at: http://www.biouml.org/sbml_tests/overview.html.

BioUML workbench version is able to simulate all 150 SBML models from BioModlels database (release 9). According to our metrics now BioUML is the best simulator for SBML models. Test details are available at: http://www.biouml.org/biomodels.shtml

1.5 Database

Modeling of biological systems requires close integration with experimental data. The distinctive feature of BioUML workbench is tight integration with biological databases. For this purpose we introduce the database type concept.

The database type defines defines:

- data types (gene, protein, RNA, substance, reaction, etc.) that are stored in the database;
- mapping of the database content into diagram elements and diagram types that can be used with the database;
- diagram types that can be used to present the database content as a set of diagrams.
- query engine to find interacting components of the system. Search results can be shown as graph and edited by user.

Databases can be installed locally or can be accessed via Internet from BioUML server.

See also

Chapter Databases

BioUML server

1.6 Search engine

BioUML workbench provides 3 types of search engine for working with databases:

- data search (filter) - this search engine maps database content into Java objects and filters these Java objects according filtering condition for each property, for example name="TP53".
- full text search - the search engine uses Lucene full text search engine. For this purpose the database content is also mapped into Java objects and then these Java objects are indexed by Lucene. Due to index usage this search engine is much faster then data search using filters.
- graph search - this search engine finds interacting pathway components and displays result as an editable graph.

Data search (filter)

This is default search engine that is available for any database. The search engine maps database into Java objects and filters these Java objects according filtering condition for each property, for example: field Complete name contains 'acid' (Figure 1.6). Regular expressions can be used for text values. This search engine works quite slowly because it scans all database objects of corresponding type (for example, gene, protein or substance).

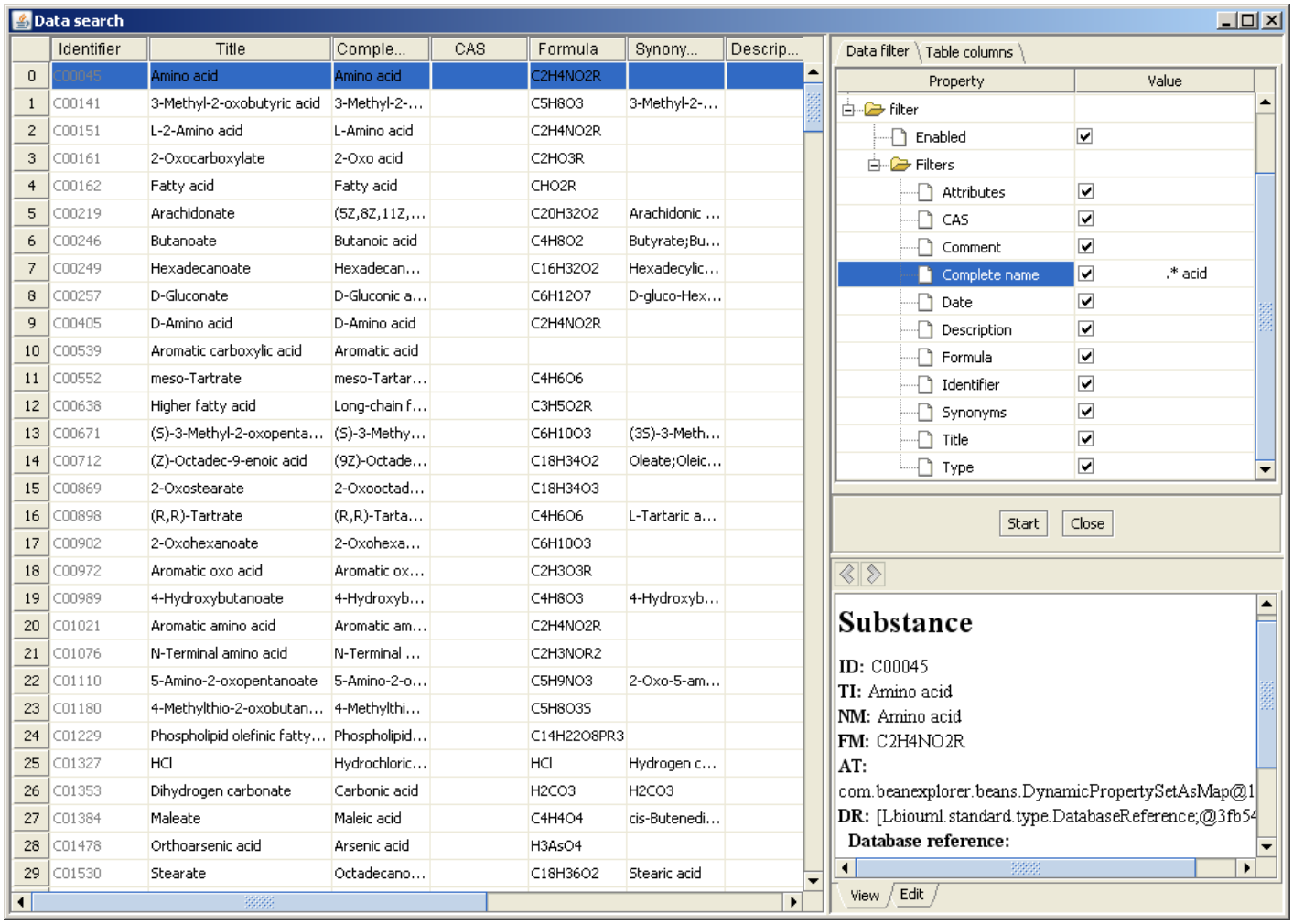


Figure 1.6. Data search dialog for KEGG/Compound, left pane - search results, top right pane - filtering conditions, bottom right pane - detailed description of substance selected in the table.

Full text search

This search engine uses Lucene full text search engine - http://lucene.apache.org/

Algorithm:

- the database content is mapped into Java objects;
- each Java object corresponds to Lucene document, object properties correspond to document fields. Administrator can specify what data types and properties will be indexed by Lucene;
- user query is parsed and executed by Lucene. Search results is set of identifiers for database objects and values of indexed fields.
- search results can be shown as a table.

BioUML workbench provides following interface for working with full text search engine:

1. Full text search dialog (Figure 1.7)
2. Full text search tab (Figure 1.8)

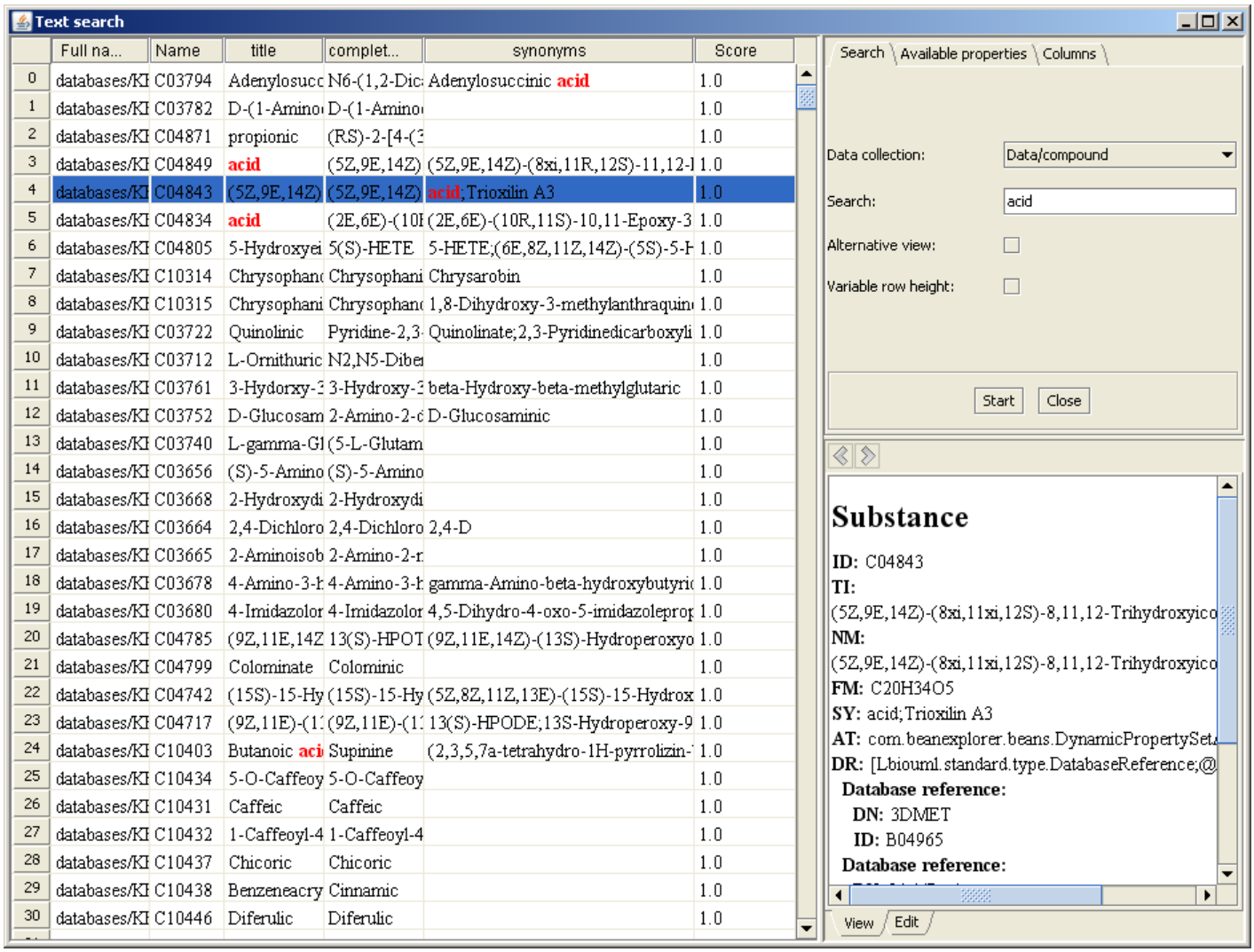


Figure 1.7. Full text search dialog for KEGG/Compound, left pane - search results, top right pane - search conditions, bottom right pane - detailed description of substance selected in the table.

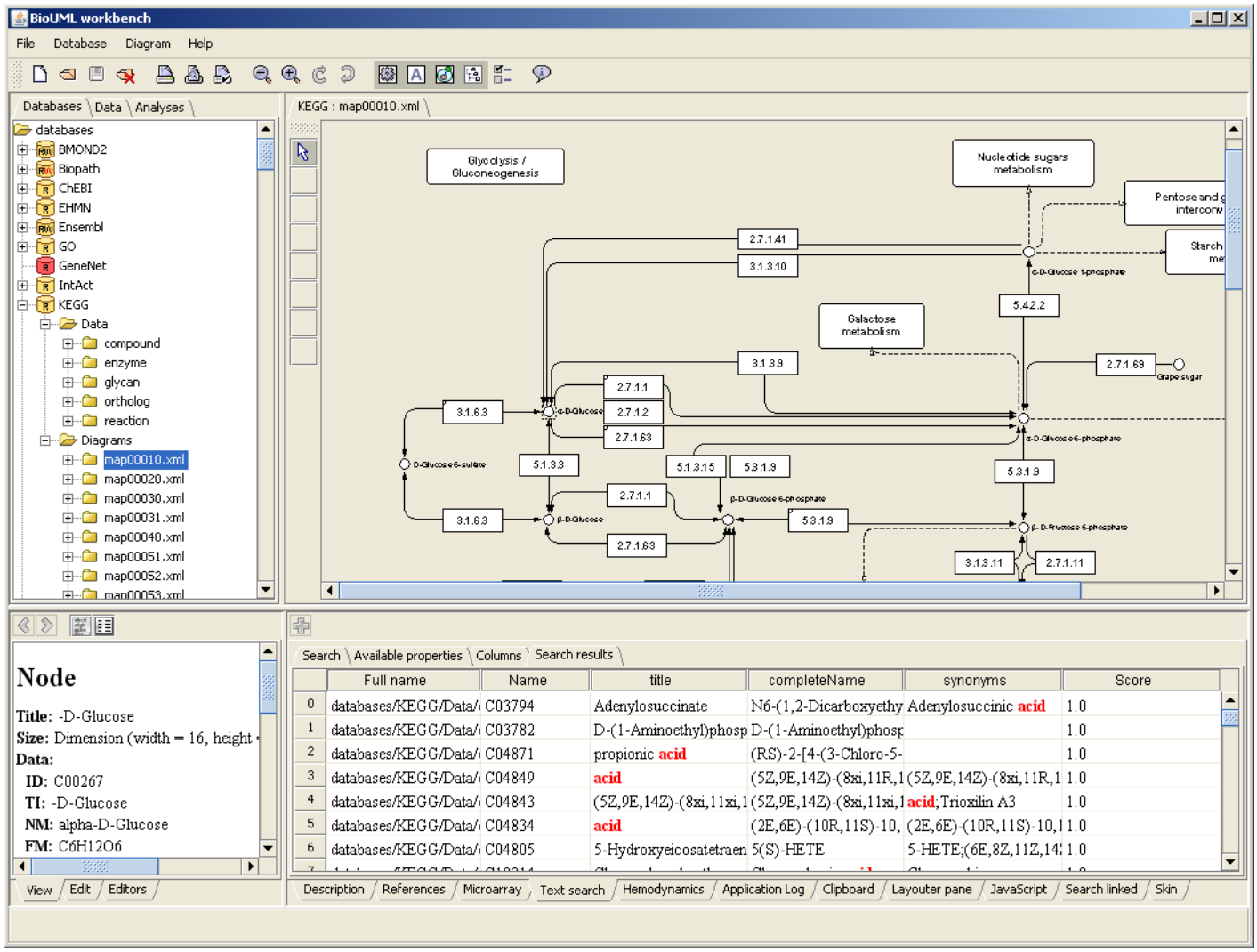


Figure 1.8. Full text search tab for KEGG/Ligand database, Compound table(bottom right pane).

Graph search

Graph search engine finds interacting pathway components and displays result as an editable graph.

To start the search user should specify start node (for example, protein AP-1 on Figure 1.9) and search conditions: what type of interactions should be found and depth of search.

Graph search supports interactive search and incremental graph layout - a user can select any node on the graph (for example, gene IL-6 on Figure 1.9) and find other biological objects in the database that interact with it. These objects will be shown in left bottom pane in tabular form. The user can add these nodes and corresponding edges to the diagram. Graph layout will locate new nodes and edges automatically preserving location of previous diagram elements (Figure 1.10).

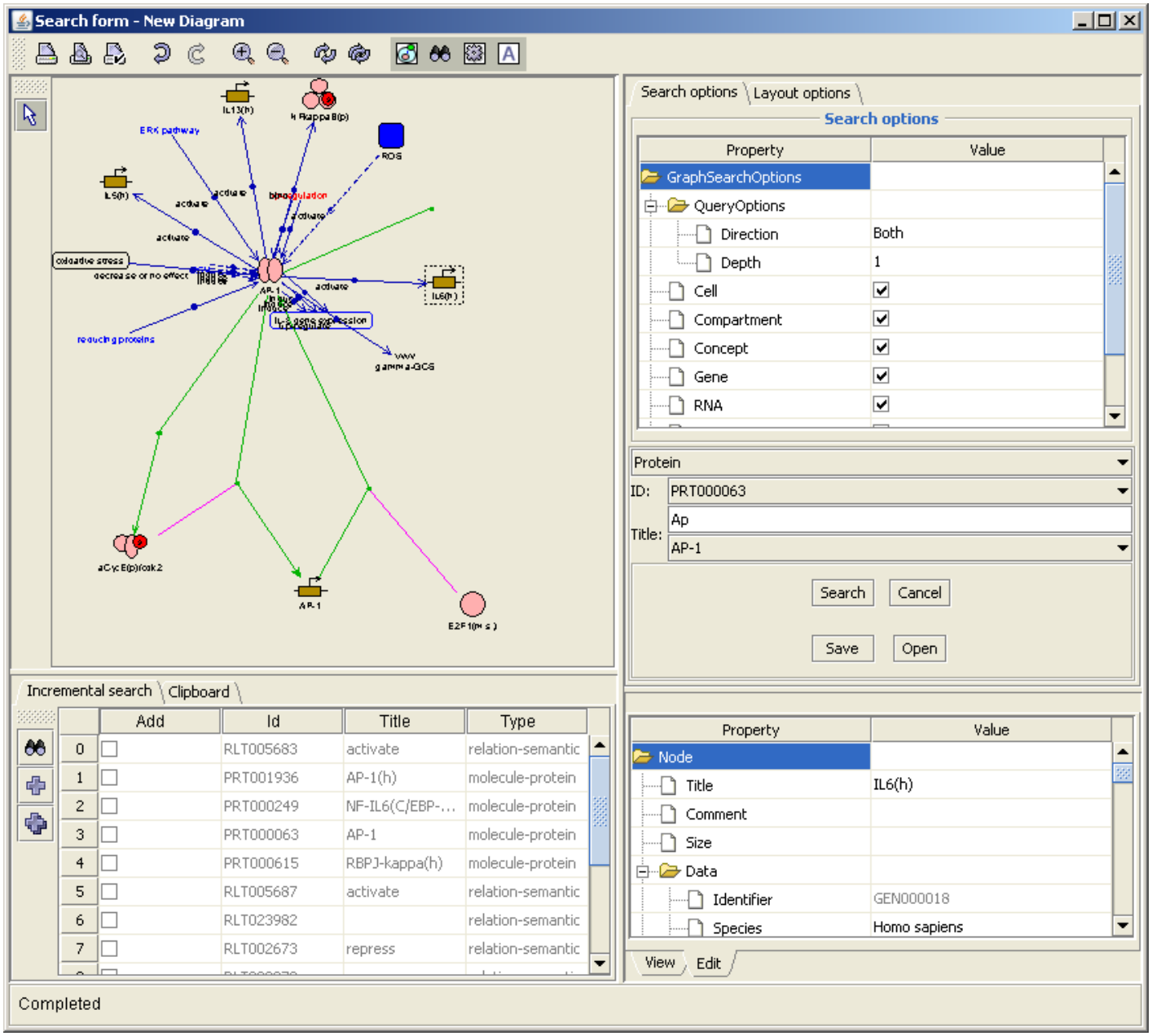


Figure 1.9. Graph search dialog. Top left pane - search results that are displayed as an editable graph, top right pane - search conditions, bottom left pane - results of interactive search for the selected node, bottom right pane - detailed description of the selected node.

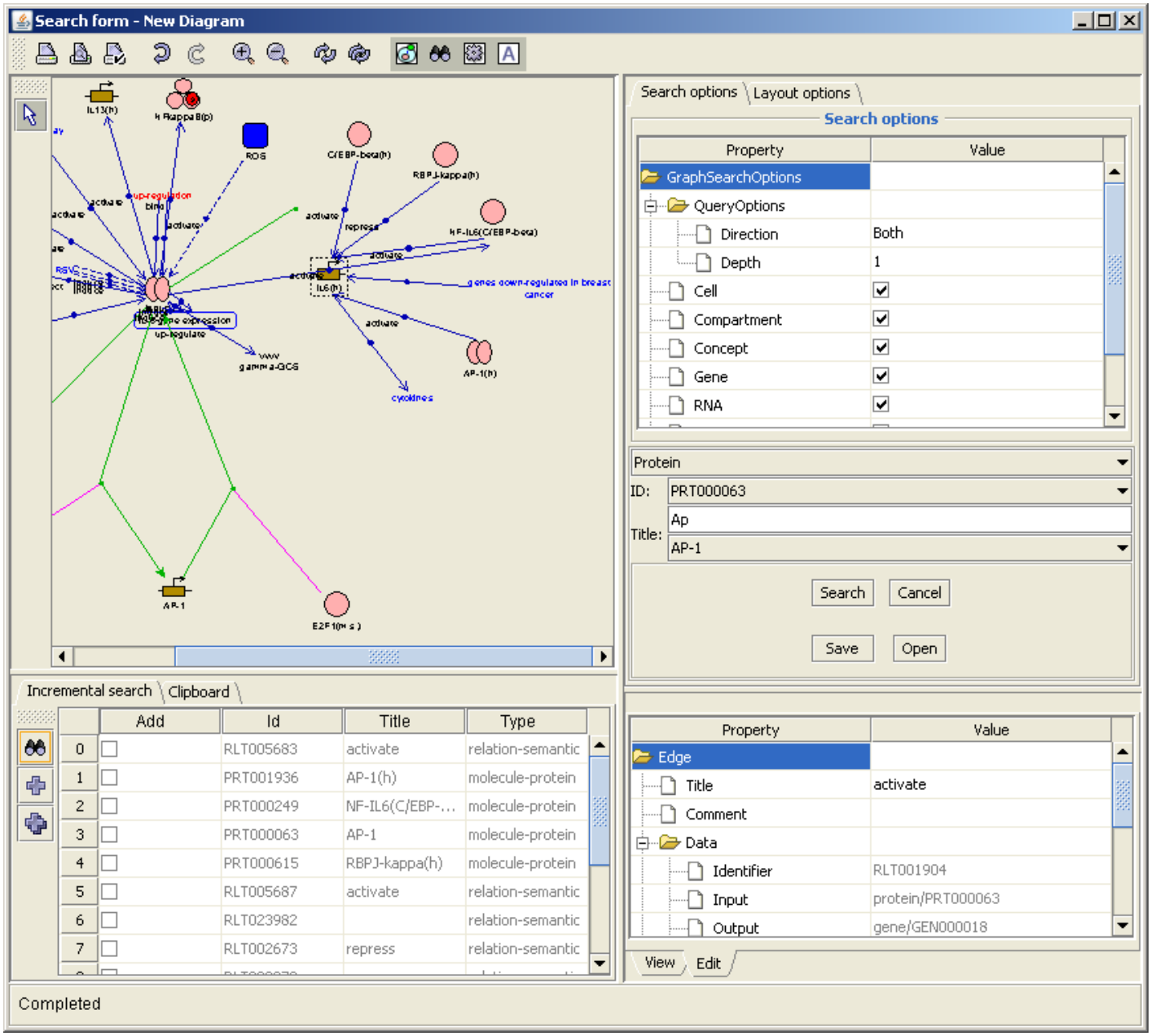


Figure 1.10. Interactive graph search, here biological objects that interact with gene IL-6 were added.

1.7 BioUML server

BioUML server provides access to databases installed on the server side for BioUML workbench via the Internet.

When BioUML workbench is started first time Setup wizard (step 3) helps a user to configure connection of BioUML workbench with BioUML server. Dialog Load databases provides the same functionality.

Security support

BioUML server supports secure access to databases. Server administrator can configure security settings for access to each database installed on the server.

User inetrface

From a user view point databases installed on the server side (remote databases) look similarly with databases installed locally and a user can use remote databases by the same way as local databases:

- a remote database content can be shown in a repository tree;
- user can add, edit and remove records from remote database;
- user can open an edit diagrams from remote database;
- all search engines (data search, full text search, graph search) are working with remote databases via special protocol;
- simulation engine can store simulation results and plots in a remote database.

Figure 1.11 demonstrates user interface provided by BioUML workbench for KEGG/Ligand database:

- repository pane (top left) - shows KEGG/Ligand database structure. Yellow icon indicates that KEGG databaseinstalled on the server and is publicly available, letter 'R' on the icon indicates that database is read only;
- diagram pane (top, right) - shows the diagram for Glycolysis/Gluconeogenesis metabolic pathway;
- object view (bottom left) - shows description of a selected node on the diagram or repository tree;
- tab for full text search (bottom right) - shows result of full text search in Compound table with query 'acid'.

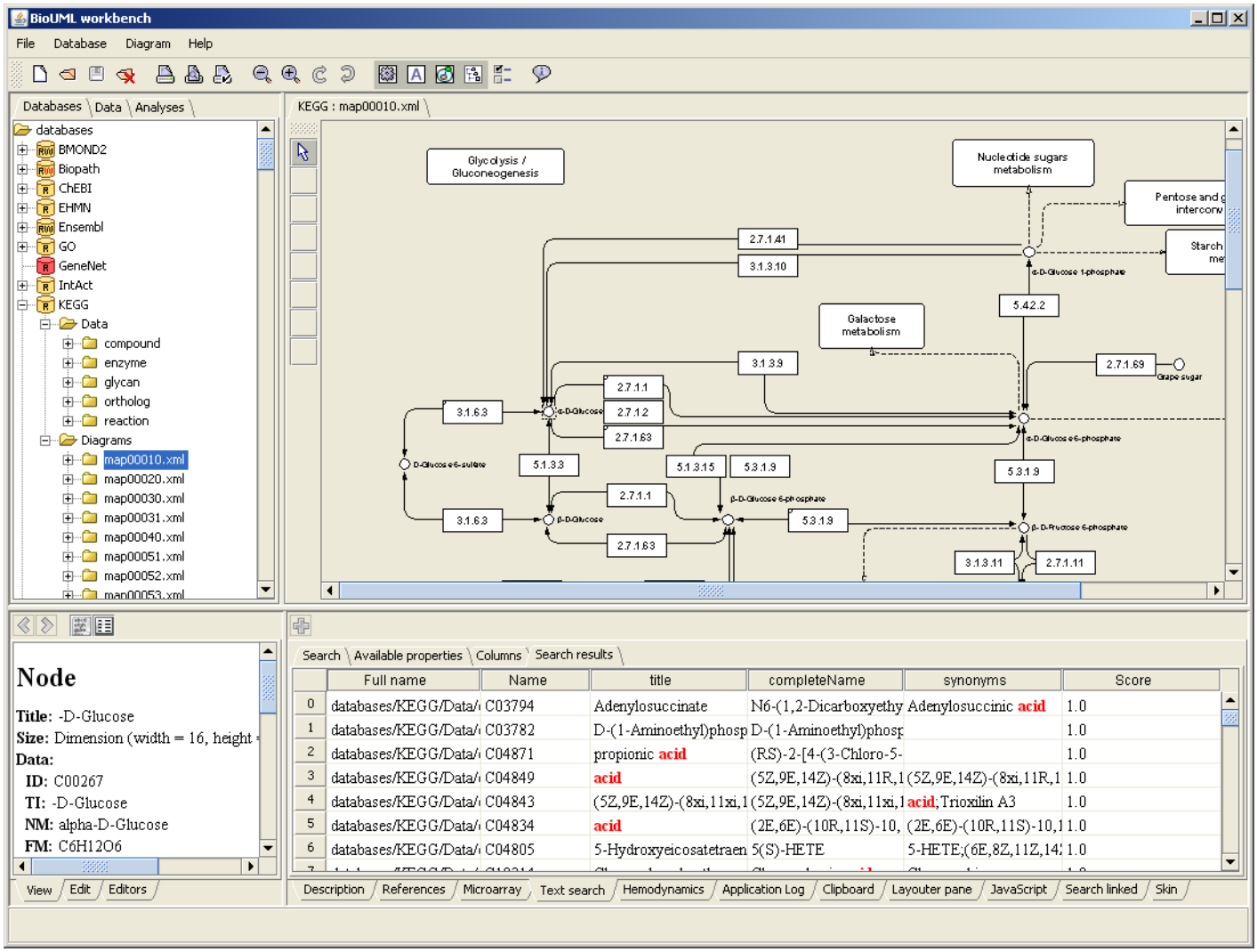


Figure 1.11. User interface provided by BioUML workbench for working with remote database (here KEGG/ Ligand database).

See also:

- Setup wizard (step 3)
- Load databases dialog
- Architecture overview

1.8 Solitaire game

Formal description and reconstruction of biological pathways is complex task. To simplify this task we introduced solitaire game as a metaphor for user interface (Figure 1.12). Here:

- the desk – a BioUML diagram
- solitaire game – reconstructed biological pathway
- cards – biological objects (genes, proteins, lipids, etc.)
- packs of cards – different biological databases (for example, Ensembl, UniProt, ChEBI, etc)

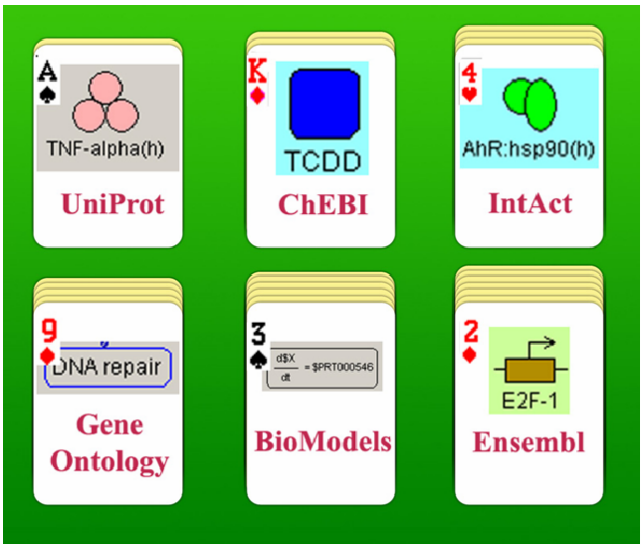


Figure 1.12. Solitaire game as a metaphor for reconstruction of biological pathways.

To support this metaphor BioUML workbench provides following functionality:

- different databases – "packs of cards" - (Ensembl, UniProt, ChEBI, IntAct, GeneOntology, Reactome, BioModels) can be installed locally or can be available via Internet from BioUML server
- convenient user interface for installation of these databases on client side (Load database dialog). For this purpose user should specify server to use and select the databases to be installed.
- composite database module – allows to specify what databases will be used for reconstruction. User can create his own database ("sandbox") where his diagrams and user added information will be stored. Composite database module is described formally using XML. Composite module editor provides user interface for its editing.
- when user add some component on a diagram he can select from which database corresponding biological object will be selected;
- full text search engine allows to search information in different databases for specified biological object type (gene, protein, etc.) using several corresponding database simultaneously. Then user can select one or more found objects and put them on diagram;
- graph search engine allows to find biologically related objects (participating in the same reaction or semantic relation) in the databases and put them into diagram.

See also:

- BioUML server
- Load database dialog

1.9 Architecture overview

BioUML workbench is a plugin-based application framework that provides its extensibility and possibility of seamless integration of other tools for systems biology. It consists from an Eclipse platform (http://www.eclipse.org) runtime kernel that supports 'plug-ins' and a set of plug-ins that support database access, diagram editing, and biological systems simulation.

Plug-in based architecture

- Plug-in - is the smallest unit of BioUML workbench function that can be developed and delivered separately into BioUML workbench. Plug-ins are coded in Java. A typical plug-in consists of Java code in a JAR library, some read-only files, and other resources such as images, message catalogs, native code libraries, etc. A plug-in is described in an XML manifest file, called plugin.xml. The parsed contents of plug-in manifest files are made available programmatically through a plug-in registry API provided by Eclipse runtime.
- Extension points are well-defined function points in the system where other plug-ins can contribute functionality.
- Extension is a specific contribution to an extension point. Plug-ins can define their own extension points, so that other plug-ins can integrate tightly with them.

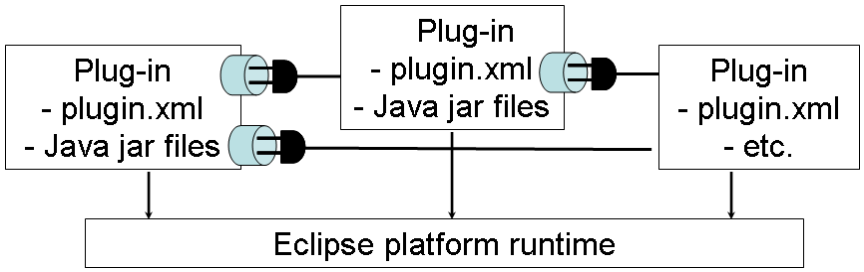


Figure 1.13. Plug-in based architecture, - extension point; - extension

Architecture of BioUML workbench

BioUML workbench installation includes a plugins folder where individual plug-ins are deployed. Each plug-in is installed in its own folder under the plugins folder. A plug-in is described in an XML manifest file, called plugin.xml, residing in the plug-in's folder. The parsed contents of plug-in manifest files are made available programmatically through a plug-in registry API provided by Eclipse runtime.

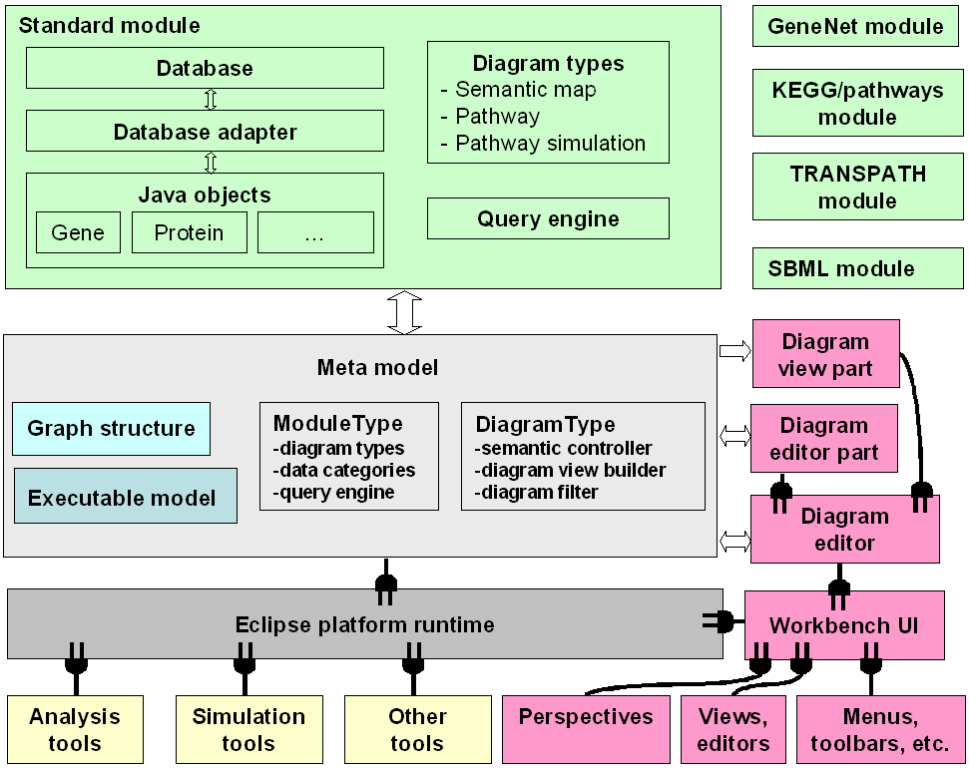


Figure 1.14. BioUML workbench - architecture overview.

Formal description and modeling of biological systems require coordinated efforts of different group of researchers:

" programmers - they should provide computer tools for this task;

" problem domain experts - they should specify what and how should be described;

" experimenters and annotators - they should describe corresponding data following to these rules;

" mathematicians - they should provide methods for models analysis and simulations.

BioUML architecture separates these tasks so they can be effectively solved by corresponding group of researchers and provides simple contract how these groups and corresponding software parts should communicate.

Architecture of BioUML server

BioUML server is Java application that is started as servlet on J2EE compatible server (we are using Tomcat server).

Like BioUML workbench it is also uses Eclipse runtime to manage by plug-ins that provide different services (Figure 1.15). Main services provided by BioUML server are:

- database service - provides information about database and secure access to it
- access service - provides access to databases (read/write)
- diagram service - provides protocol to read/write diagram and all diagram elements during one HTTP request
- Lucene service - provides full text search and its configuration
- query service - provides indexed search for a database

BioUML server supports access to different types of databases. Main of them are:

- relational databases (for example, Ensembl database that is available as MySQL dump)
- text databases (for exampe KEGG/Ligand database)
- XML databases (for example databases in BioPAX or SBML formats)
- databases available via web services (for example SABIO-RK database)

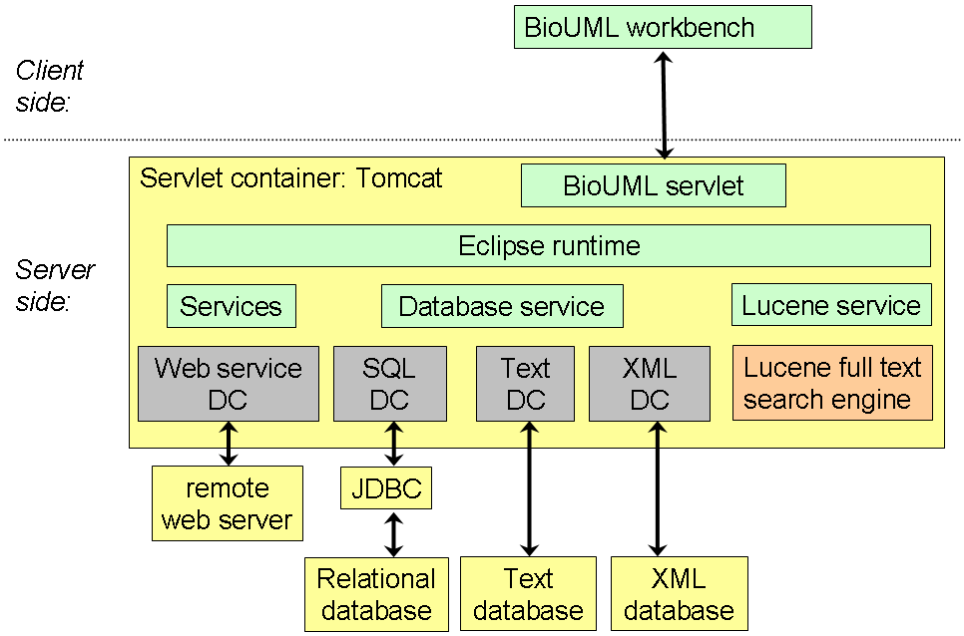


Figure 1.15. Architecture of BioUML server.

2 Installation and configuration

2.1 Installation

Hardware and Software Requirements

Hardware

- Pentium II 500 or higher.
- 512 MB RAM.
- 50 MB free hard disk drive space.

Software

- Java virtual machine - JDK 1.6

You may download JDK 6 from Sun Microsystems Inc web site
http://java.sun.com/javase/downloads/?intcmp=1281

BioUML Installer

You can download the latest version of BioUML workbench from BioUML web site:
http://www.biouml.org/download.shtml?download

To start BioUML workbench installer type in command line:
java -jar BioUML-install-yyyy_mm_dd.jar

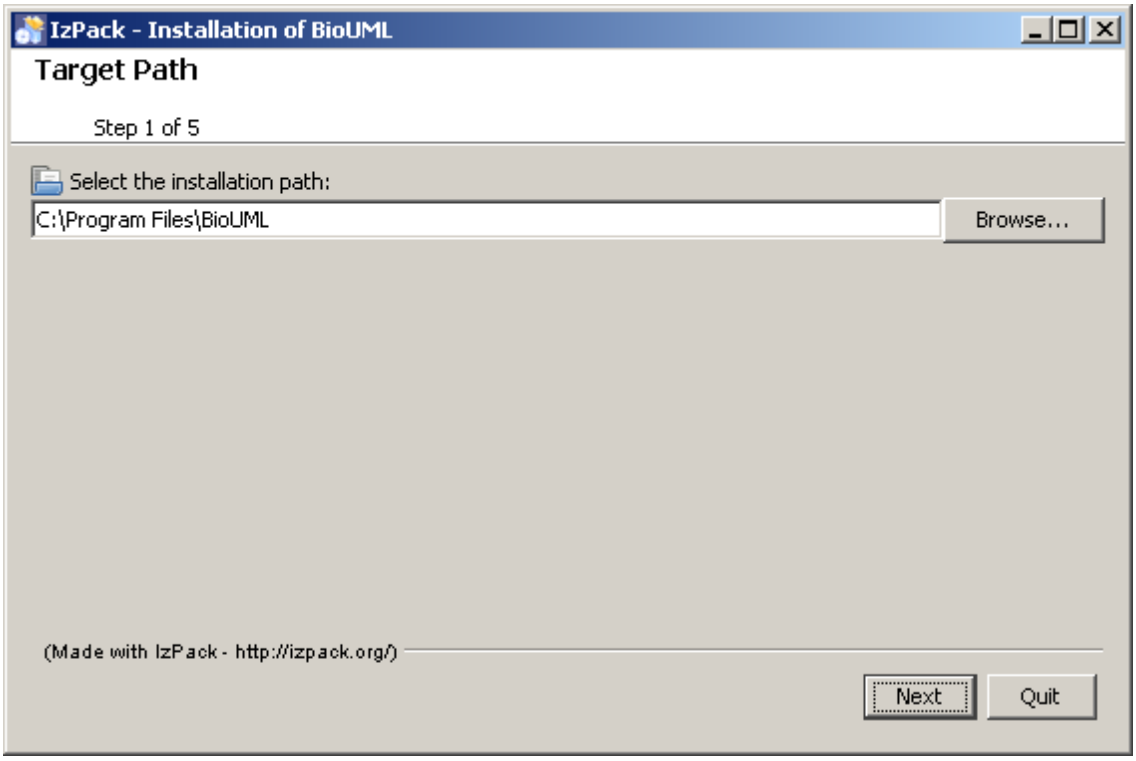


Figure 2.1. BioUML installer, step 1.

BioUML installer will guide you through the installation process (Figure 2.1). The installation process is straightforward and consists from 5 steps:

1. Target path - specify a directory where BioUML workbench will be installed.
2. Select installation packages - here you can see list of BioUML plug-ins to be installed. We recommend to select all (the default behaviour).
3. Installation - installation progress.
4. Set up shortcuts - allows you to create shortcuts in the Start menu and on the desktop to start BioUML workbench.
5. Installation Finished.

After successful installation you can start BioUML workbench. BioUML setup wizard will help you to configure BioUML workbench and install remote databases from BioUML server.

See also

Setup wizard

Updates

2.2 Setup wizard

Setup wizard will help you to configure BioUML workbench and configure connection with remote databases on BioUML server.

Setup wizard is started automatically during first and further runs of BioUML workbench till you uncheck **Show on startup** check box.

You can also start Setup wizard from menu: **File > Setup wizard**.

The wizard will guide you through 3 steps:

1. **Proxy settings** (Figure 2.2) - you should specify HTTP proxy server settings. This is essential if you would like to work with remote databases from BioUML server and your network is protected by firewall.

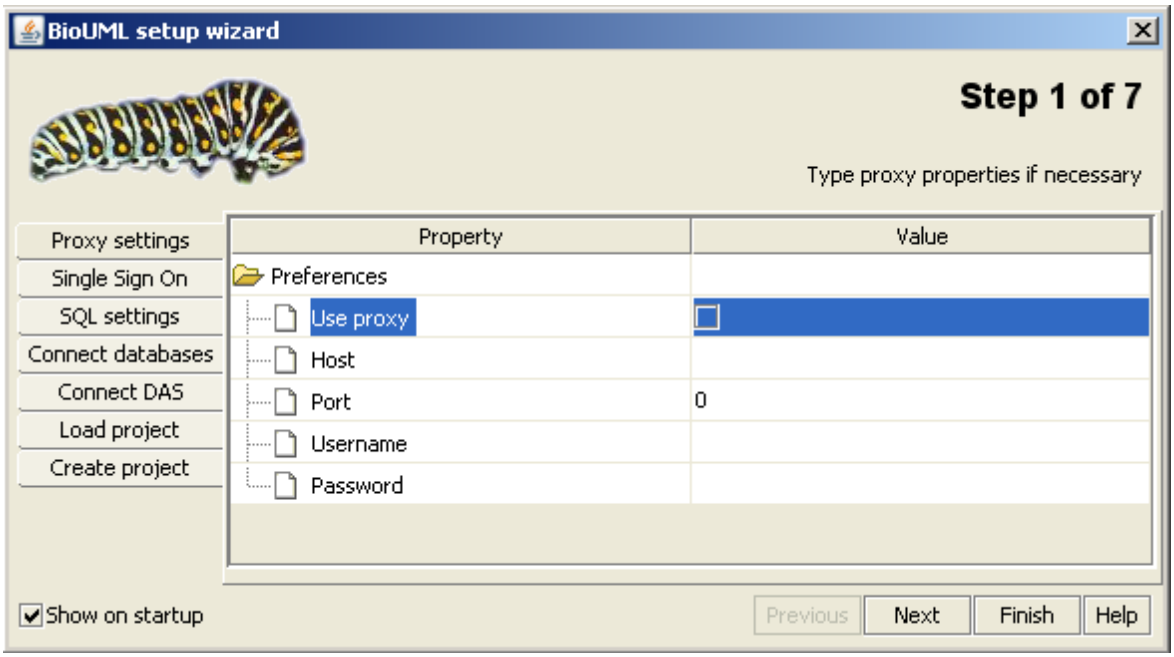


Figure 2.2. Setup wizard, step 1 - specifying settings for HTTP proxy server.

2. **Single Sign On** (Figure 2.3) - this mechanism allows you to login into BioUML workbench only once. Then BioUML workbench will store and use all your logins and passwords for access to protected databases on the BioUML server. This information is stored in encrypted form.

   Check **Use single sign on** checkbox if you would like to use single sign on mechanism. After that you should specify your **User name** and **Password**. During next run BioUML workbench asks you this information for log in (Figure 2.4).

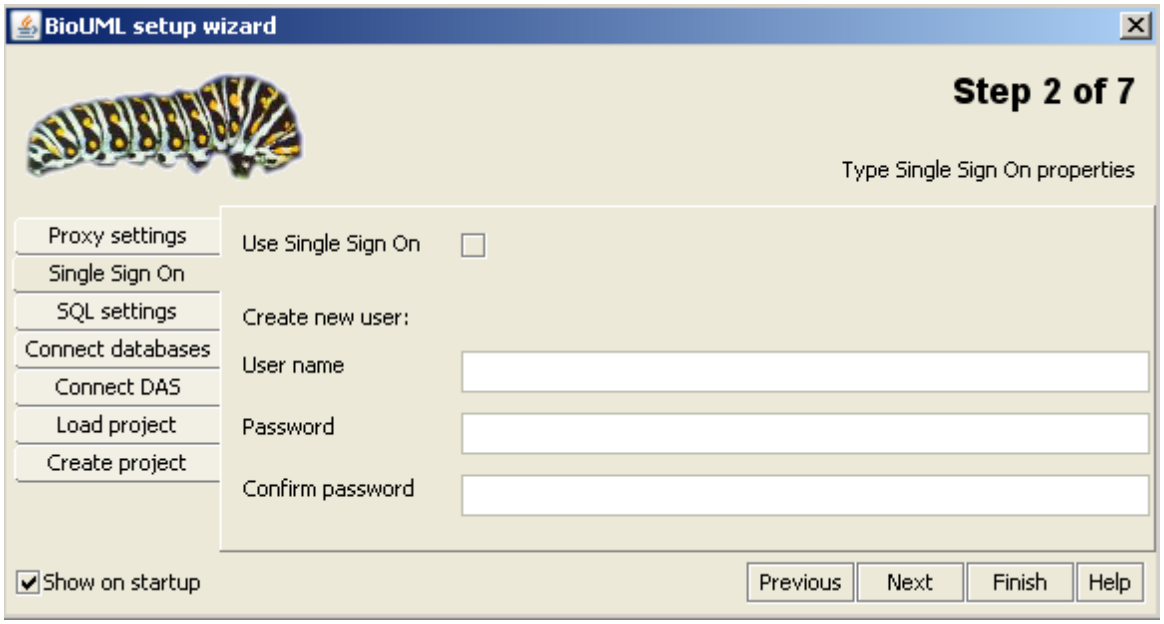


Figure 2.3. Setup wizard, step 2 - configuring Single Sign On.

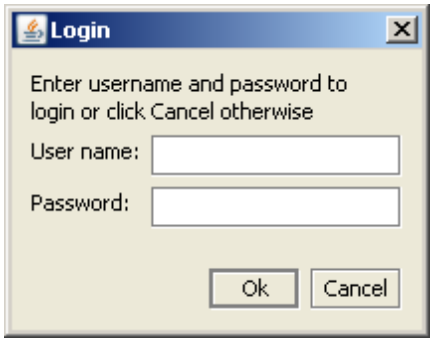


Figure 2.4. Login dialog, it is shown during starting BioUML workbench if Single Sign On is used.

3. **SQL settings** (Figure 2.5) - allows user to configure SQL connections. SQL connections can be used to save data into SQL database (for example, tables)

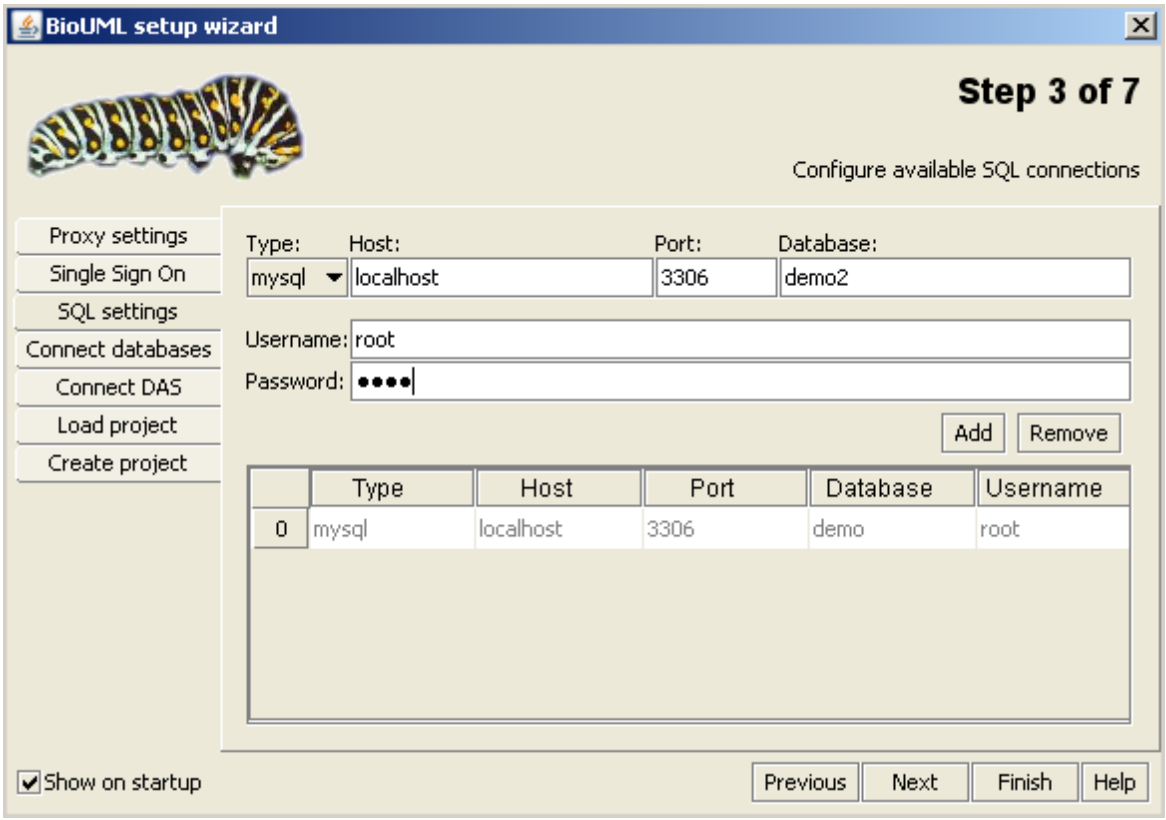


Figure 2.5. Setup wizard, step 3 - configuring SQL connections.

4. **Connect databases** (Figure 2.6) - helps a user to configure connection of BioUML workbench with BioUML server for access to remote databases. This step of the wizard provides the same user interface as Load database dialog where you can find more details.

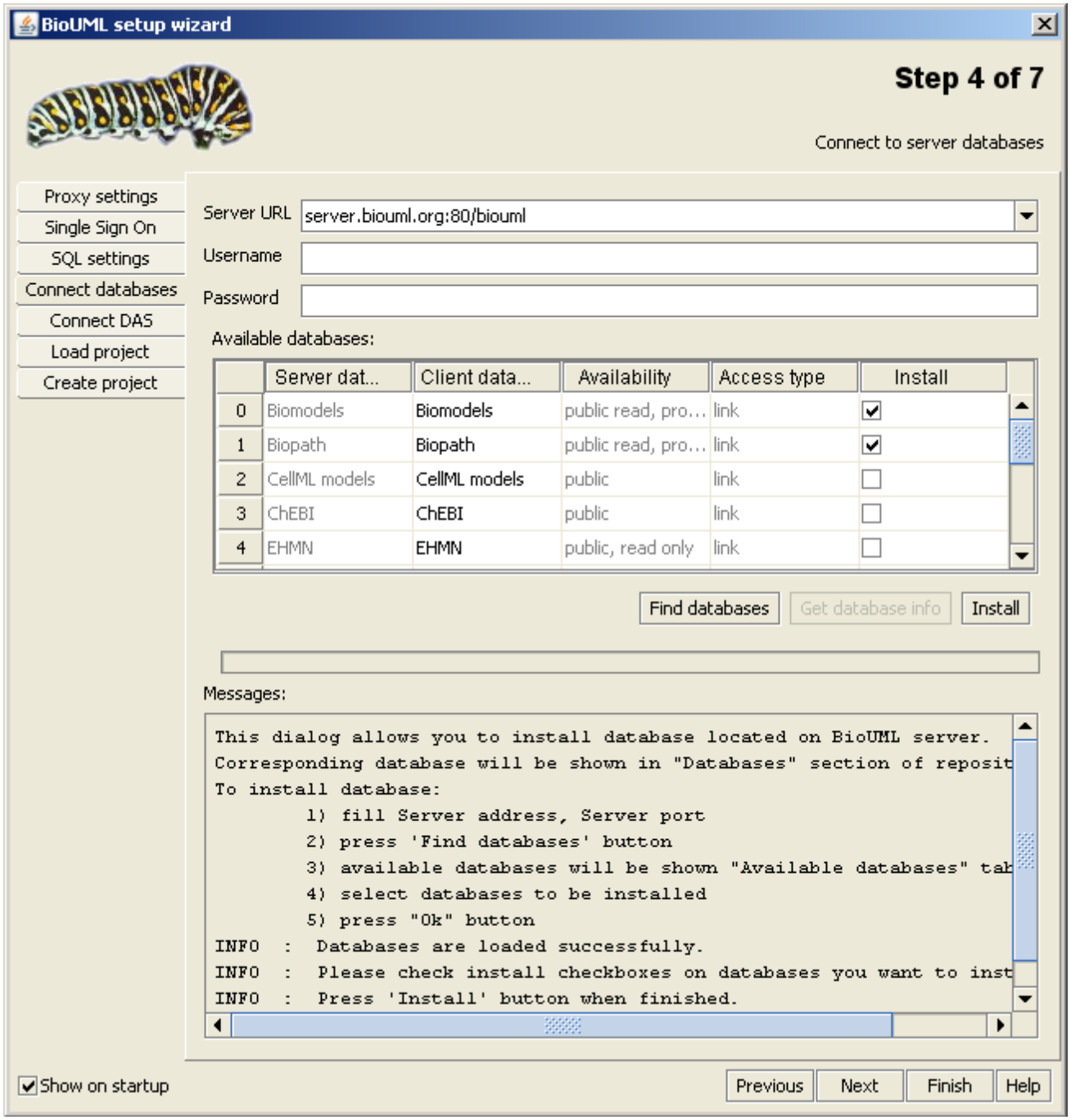


Figure 2.6. Setup wizard, step 4 - configuring connection with remote databases on a BioUML server.

5. **Connect DAS** (Figure 2.7) - helps a user to configure connection of BioUML workbench with remote DAS servers. See also: Import DAS

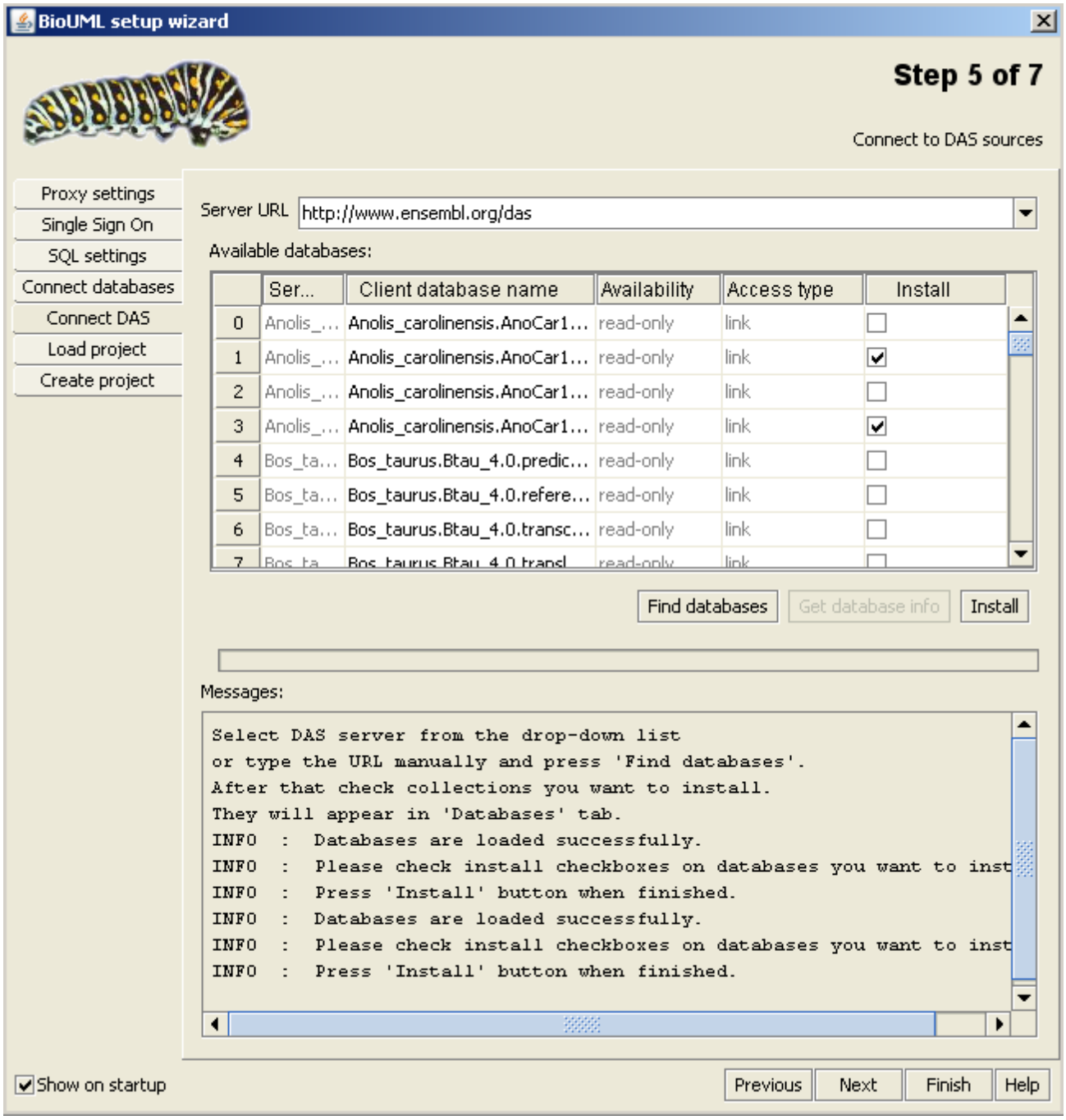


Figure 2.7. Setup wizard, step 5 - configuring connection with remote DAS servers.

6. **Load project** (Figure 2.8) - helps a user to load projects from BioUML server.

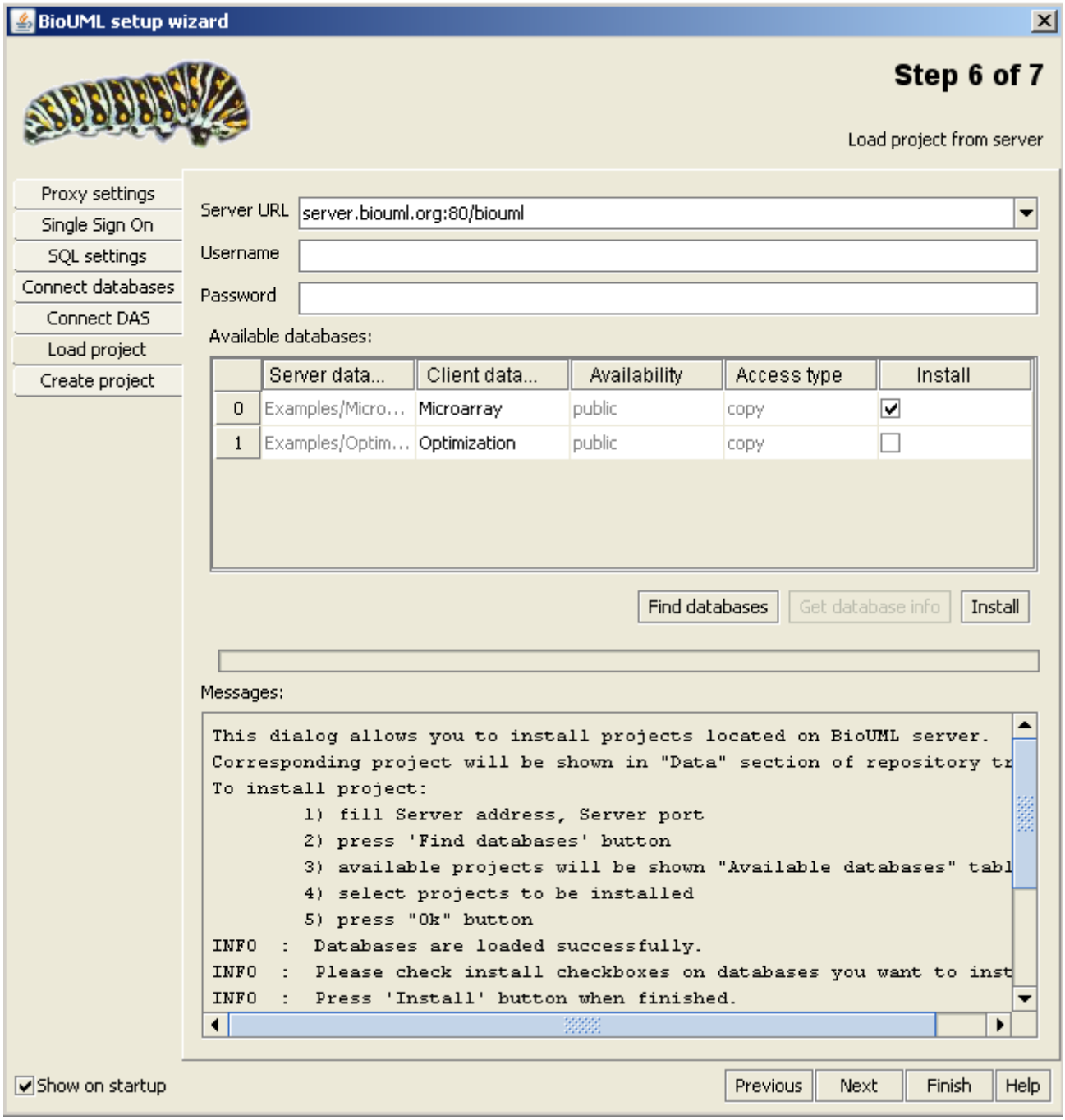


Figure 2.8. Setup wizard, step 6 - load projects from BioUML server.

7. **Create project** (Figure 2.9) - allows to create new research projects.

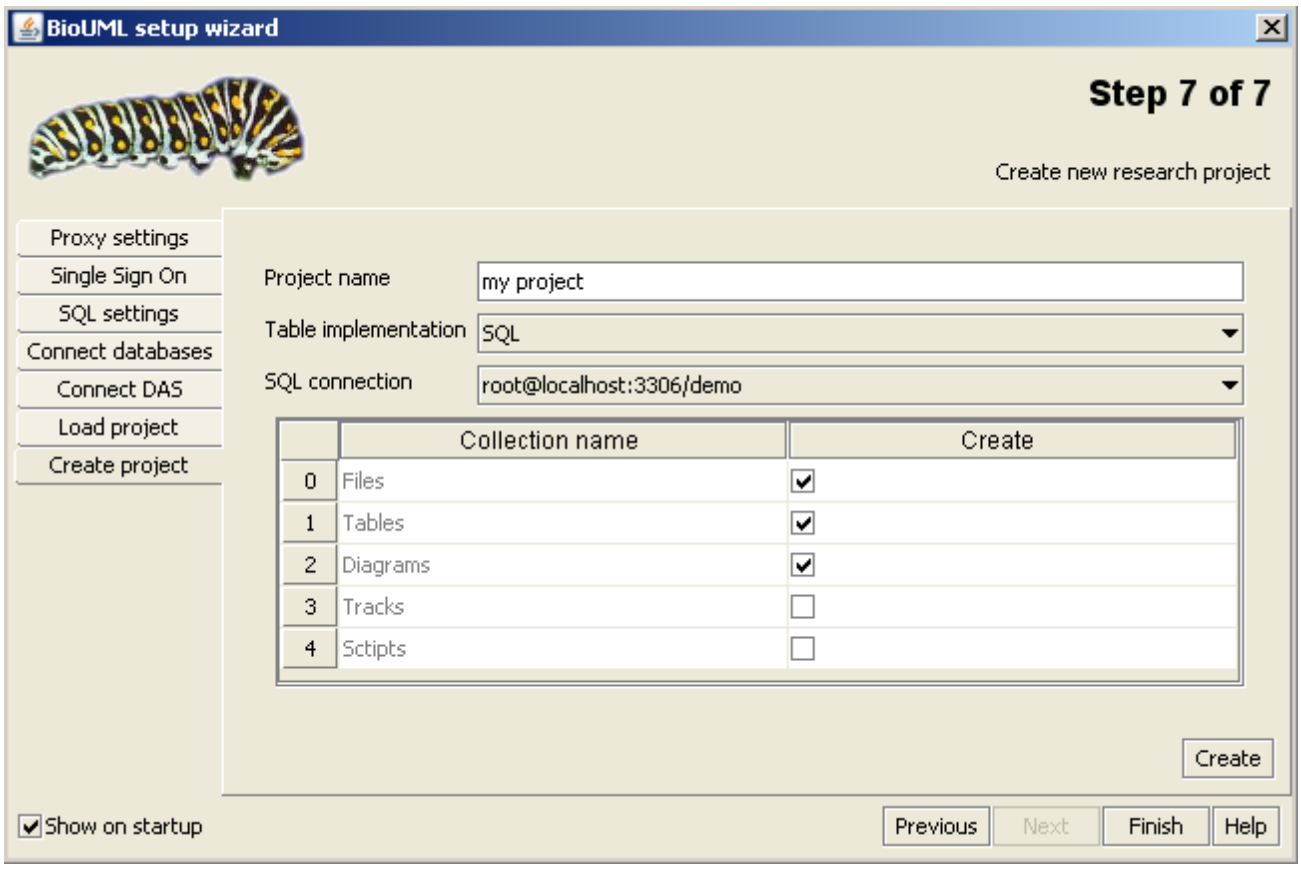


Figure 2.9. Setup wizard, step 7 - create new projects.

See also:

- BioUML server
- Load database dialog

2.3 Updates

BioUML workbench provides automatic updates of installed plug-ins by newer versions from the specified update server.

3 Databases

Modeling of biological systems requires close integration with experimental data. The distinctive feature of BioUML workbench is tight integration with biological databases.

Databases can be installed locally or can be accessed via Internet from BioUML server. BioUML server supports secure access to databases. Server administrator can configure security settings for access to each database installed on the server. Information where database installed (locally or on the server side) as well as about its availability is displayed using different icons (Figure 3.1).

Icon color indicates where database is installed and its accessibility:

- blue color - database is installed locally, accessible for reading and writing;
- yellow - remote public database, accessibility for reading and writing is specified by R and W letters;
- red color - remote protected database, user should log-in to get access to the database, accessibility for reading and writing is specified by R and W letters;
- green color - remote protected database, user successfully logged in, accessibility for reading and writing is specified by R and W letters;

Table 3.1.

Icons for local and remote databases

Local database

local database, available for reading and writing

Remote database

public remote database, read only

public remote database, available for reading and writing

before log-in		*after log-in*
	remote public database, requires log-in for writing	
	remote protected read only database, requires log in for reading	
	remote protected database, requires log in for reading and writing	

Figure 3.1. Icons for local and remote databases.

See also:

- BioUML server

3.1 User interface

User interface for access to databases consists of:

- repository pane that shows the database content (Figure 3.1, top);
- property inspector that shows information about selected node in repository pane (Figure 3.2, bottom). Property inspector has 2 tabs:
  - view - shows information about selected node in repository pane as HTML text
  - edit - allows a user to edit information about selected node
- search engines - 3 types of search engine can be used working with databases: data search by filter condition, full text search and graph search for
- Load database dialog - helps a user to configure connection of BioUML workbench with BioUML server for access to remote databases.

See also:

- Search engine
- Load database dialog

Repository pane and property inspector

List of available databases is shown in repository pane in tab **Databases** (Figure 3.1).

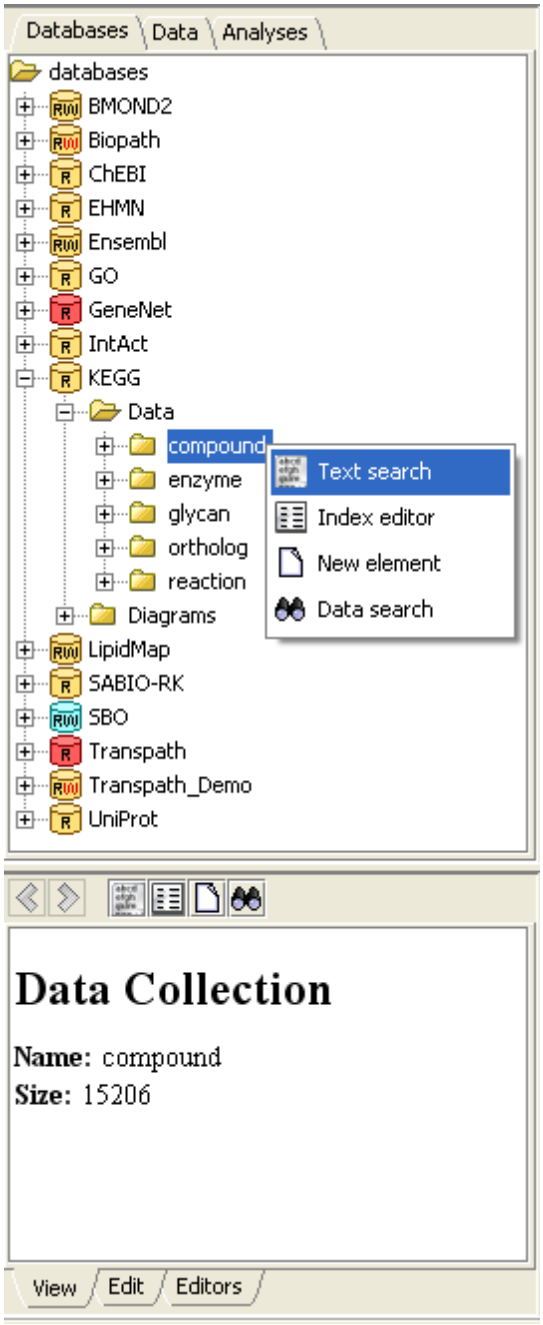


Figure 3.1. Repository pane (top) and property inspector (bottom).

3.2 Load database dialog

Load database dialogs (Figure 3.1) helps a user to configure connection of BioUML workbench with BioUML server for access to remote databases. Setup wizard, step Load databases provides the same user interface.

From a user view point remote databases installed on the server side look similarly with databases installed locally, so we may call this process as installation or loading of remote databases.

To install remote databases using **Load database dialog**:

1. Select from menu **Database > Load database** item. Load database dialog will be opened (Figure 3.2).

2. Specify **Server URL** for connection with BioUML server or you can use default BioUML server.

3. Enter **Username** and **Password** for server authorization (use blank values for guest connection)

4. Click **Find databases** button. All databases installed on the specified BioUML server will be shown in table **Available databases**.
   The table columns are:
   - Server database name - name of the database on BioUML server
   - Client database name - name of the database how it will be shown in the repository tree. By default database name on the client side is the same as on the server side, however a user can change it. For example user can add version of the database or server name as the suffix.
   - Availability - describes the availability of the database:
     - public - database is publicly available for reading and writing;
     - public, read only - database is publicly available for reading only;
     - public read, protected write - database is publicly available for reading, for writing user should log in only;
     - protected - user should log in to read/write information from the database;
     - protected, read only - user should log in to get access to the database, read only
   - Access type - either 'link' (database will be linked and accessed remotely) or 'copy' (database will be copied from the server and accessed locally)

5. To get information about a database:
   - select the database in the table by clicking on corresponding raw;
   - press **Get database Info** button;
   - information about the database will be shown in **Messages** pane (Figure 3.3).

6. Select in **Install** column databases to be installed by clicking on corresponding check box.

7. Press **Install** button. Information about installation process will be shown in **Messages** pane.

8. Press **Close** button to close the dialog after successful installation of remote databases.

Notes:

1. The database can be installed only once. If you will select the same database to be installed again the it will be skipped and corresponding message will be shown in Messages pane, for example:
   ```
   WARN  :  Database with the same name already exists ('ChEBI')
   ```

2. Some databases (complex databases) include information from other databases. During the installation BioUML workbench automatically checks such dependencies and suggests to install required databases (Figure 3.4). Press **Ok** button to install required databases too.

3. Some databases may require additional plug-ins that are not included into default distributive of BioUML workbench.
   Public version of BioUML workbench **do not** include plug-ins for following databases:
   - TRANSPATH;
   - GeneNet.

See also:

- BioUML server
- Setup wizard

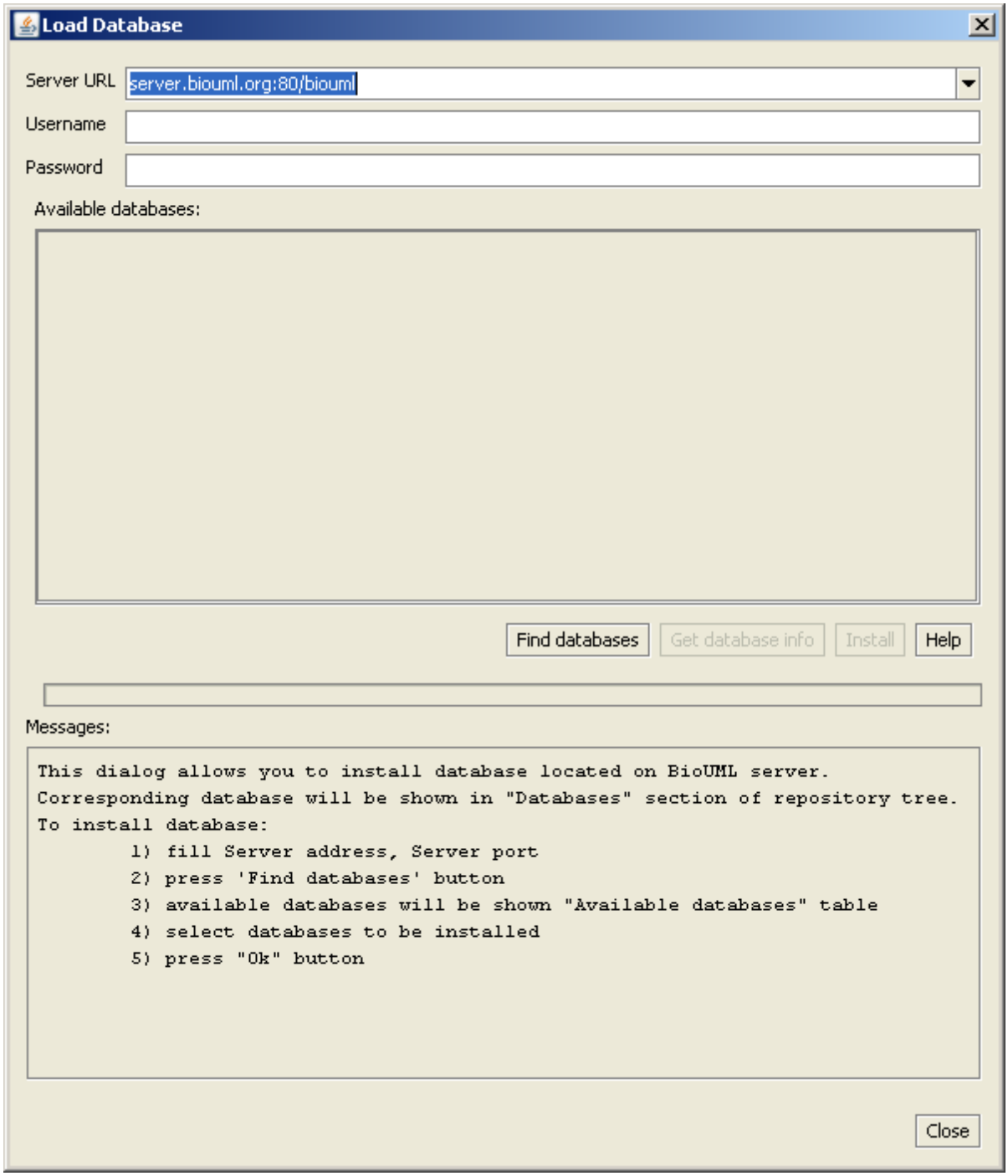


Figure 3.2. Load database dialog.

```
INFO  :
Database: Biopath
Version: 0.8.5
Upadate: 05.02.2009
Availability: public read, protected write
Description: Test description for Biopath database
Statistics:
```

```
Data:
      cell: 84
      compartment: 220
      concept: 2388
      gene: 442
      literature: 1533
      protein: 3561
      reaction: 4838
      relation: 25008
      rna: 38
      substance: 4894
Diagrams: 555
Dictionaries:
      database info: 30
      relation type: 2
      species: 9
      unit: 2
Simulation: 2
```

Figure 3.3. Example of information about the database that is shown in Messages pane.

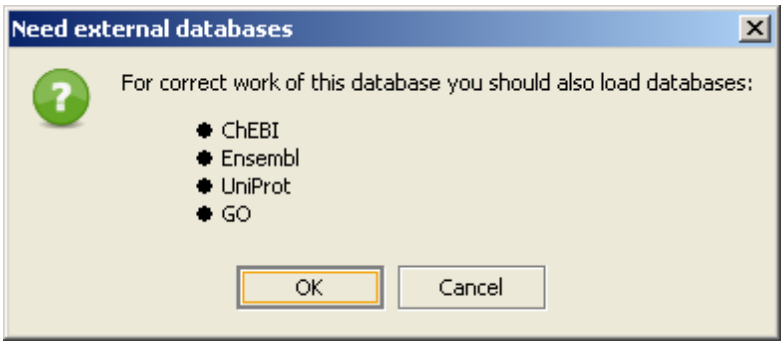


Figure 3.4. Dialog that shows dependency of installed database from others.

3.3 Import database

BioUML extensions provides database import from external sources, such as BioPAX, OBO files or external DAS-server.

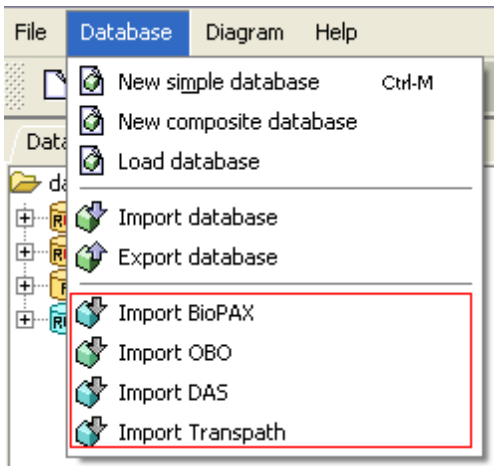


Figure 3.7. Import external database menu.

Supported source types:

- Import DAS (provided by DAS plugin)

3.3.1 Import DAS

Import DAS action allows to create new database based on external DAS-server data.

NOTE: Before DAS importing check your proxy server settings in BioUML Preferences dialog

In Import DAS dialog enter DAS server name and press **Find databases...** button. The list of available DAS sorces will be shown in table below (Figure 3.8)

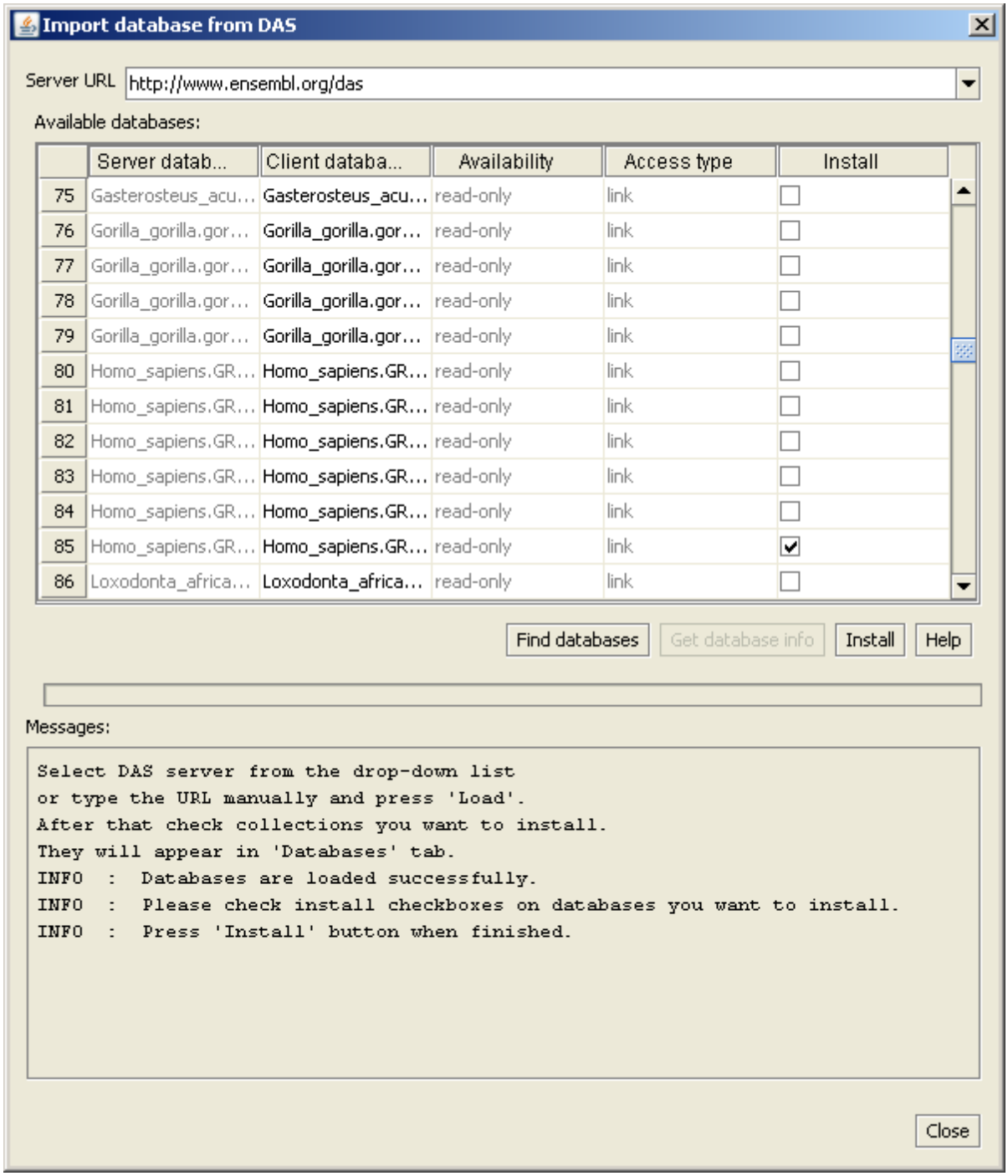


Figure 3.8. Import DAS dialog.

You can select DAS sources you want to install by checking the checkboxes in the **Install** column. Optionally you can specify arbitrary name for new database in **Client database name** column. You may read additional database description by pressing **Get database info** button while source is selected. Finally press **Install** button to install selected sources as BioUML databases.

New databases will be created and available from repository tree (Figure 3.9)

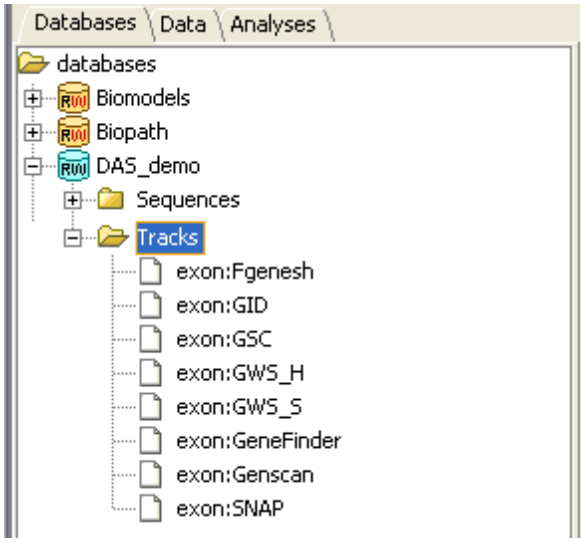


Figure 3.9. DAS database in repository tree.

3.4 Text search

Text search provides database elements search by indexed element fields. In most case element can be found by name, title, description and other specific fields.

Search pane at the bottom of database tree is the simplest way to use search functionality. Select database or its subcollection in **databases** tree, enter search string in search pane text field and click **Search** button on the search pane(Figure 3.5).

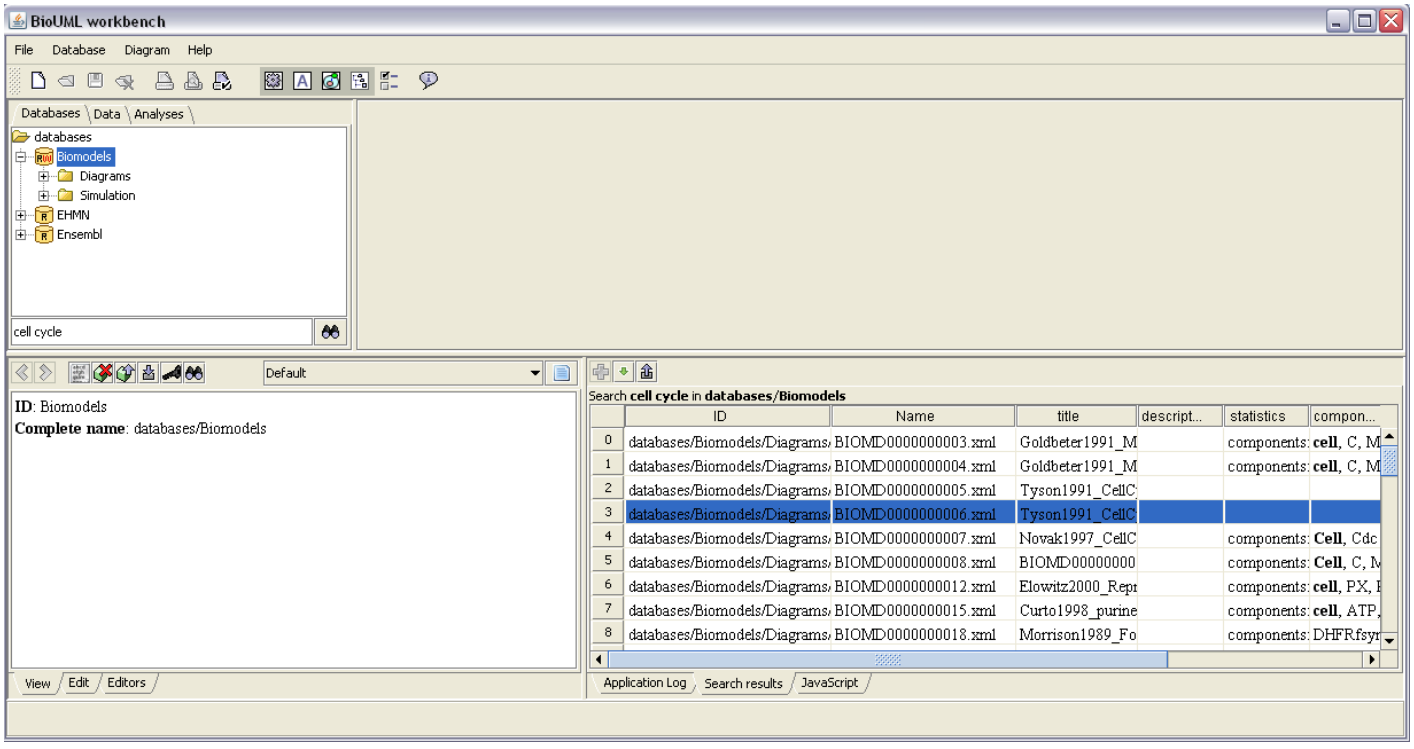


Figure 3.5. Using search pane.

The results will be able from **Search results** view part at right bottom part of BioUML application.

Search results actions.

Search results view part provides some actions which are able from its toolbar.

- Add element
- Full mode
- Export search result

3.4.1 Add element

Add new element on the diagram action is able if current opened document is diagram. Select element in search result table and click **Add new element on the diagram** action. After that check the location of the new element on the diagram by mouse left click. If diagram supports this type of elements you'll see new element on the diagram or error message otherwise.

3.4.2 Full mode

Set full mode of search result table. Click one more time to return to compact mode.

Simple mode

The height of each row is fixed. The table is compact but some text is hidden.

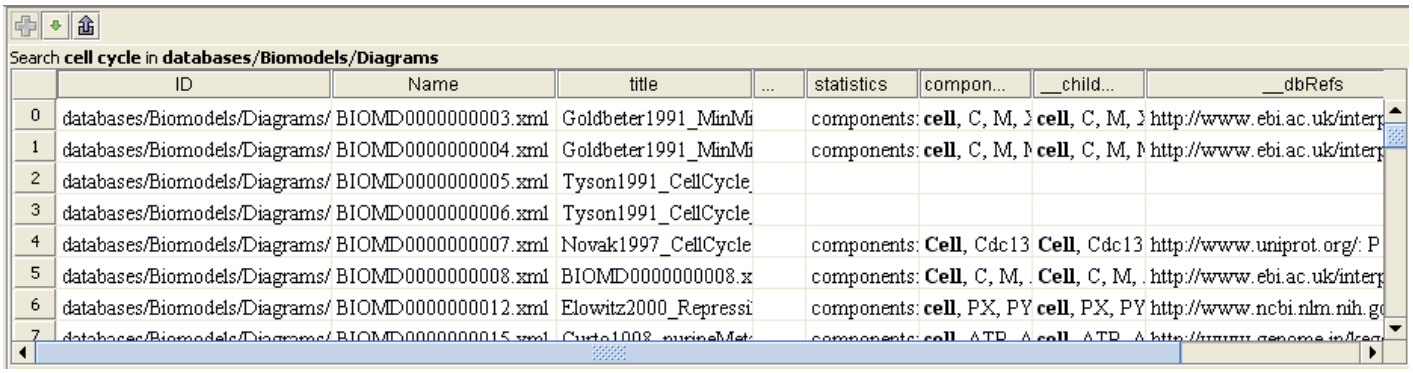


Figure 3.6(a). Simple mode of **Search results** pane.

Full mode

The height of each row is enough for cell content.

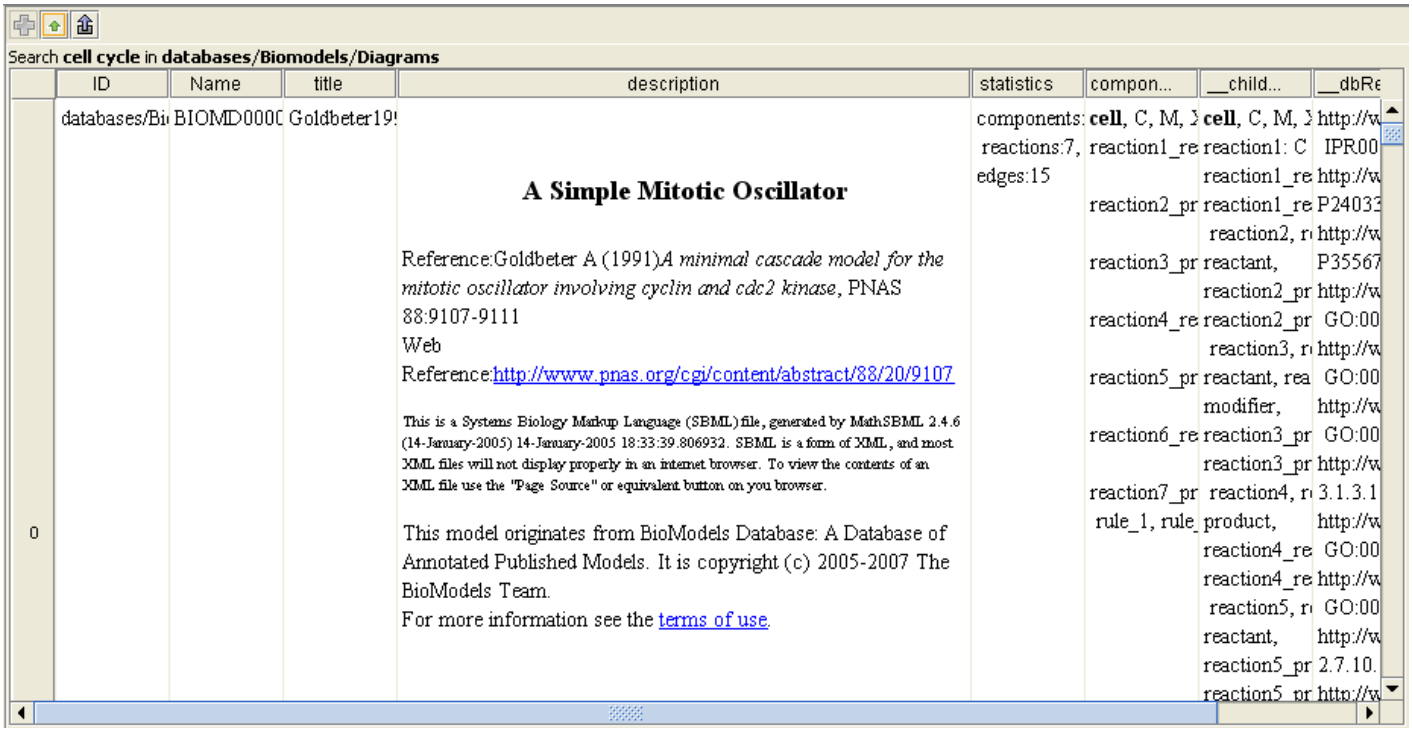


Figure 3.6(b). Full mode of **Search results** pane.

3.4.3 Export search result

Export table to file. In special dialog you can select format, export columns and rows and set compression mode (Figure 3.6).

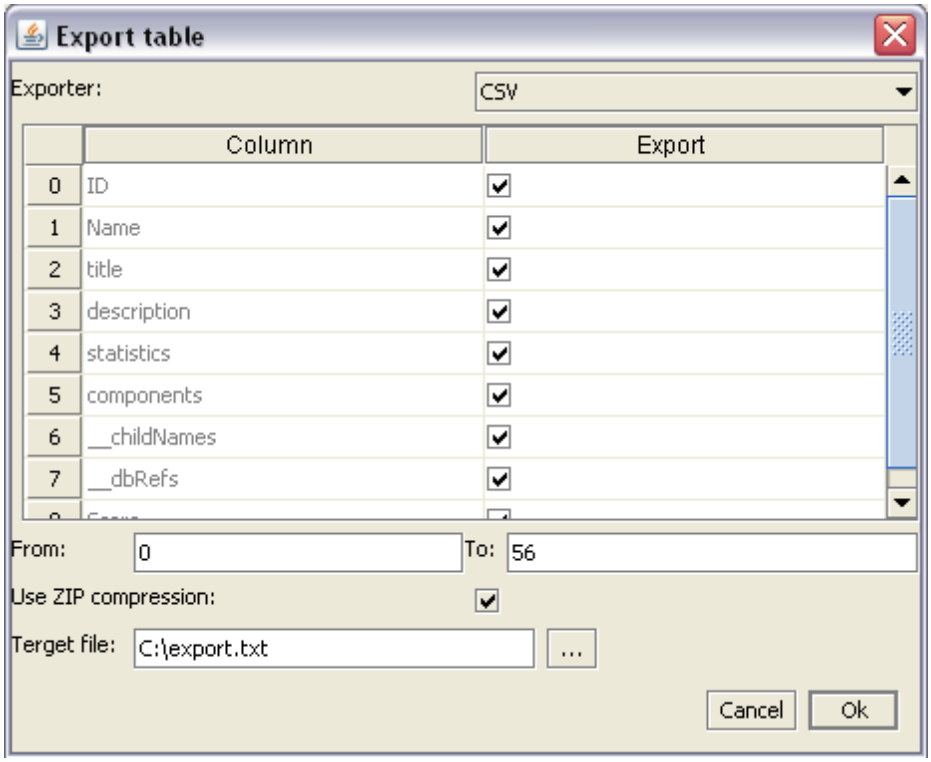


Figure 3.6. Export search results table.

4 Diagram types

See also:

- Diagram type
- Chapter "Graphic Notation Editor"

4.1 Standard diagram types

BioUML workbench defines 5 standard diagram types that allows a biologist to describe biological pathways (metabolic pathways, signal transduction pathways and gene networks) on different level of abstraction and with different level of details (Figure 4.1.).

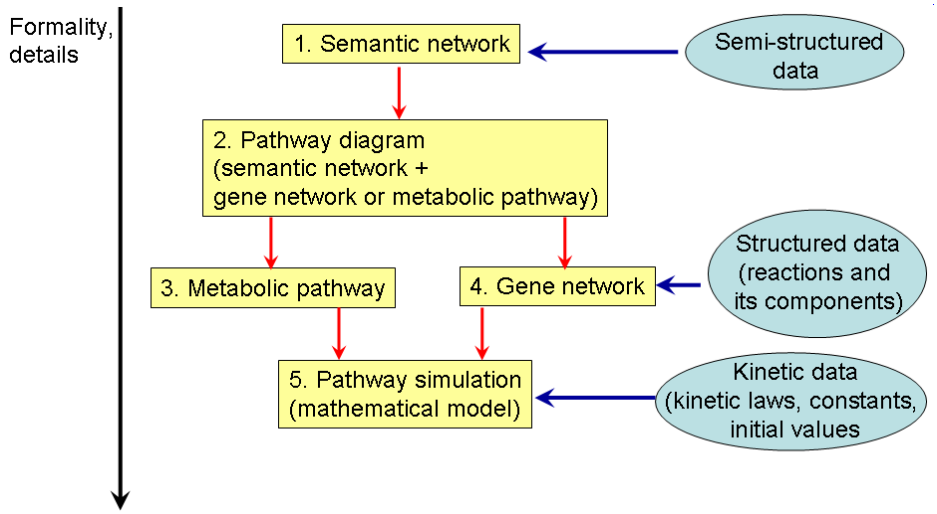


Figure 4.1. Reconstruction and formal description of biological systems using different diagram types.

4.2 SBGN

Despite having one of the highest ratios of graphical to textual information, biology still lacks standard graphical notations.

Recently Systems Biology Graphical Notation (SBGN), a visual language developed by a community of biochemists, modelers and computer scientists was suggested (Le Novère et al., 2009).

SBGN consists of three complementary languages: Process Diagram, Entity Relationship Diagram, and Activity Flow Diagram.

5 Diagram layout

BioUML provides automatically diagram layout feature.

Open diagram and select **Layout** view part at the right bottom pane of BioUML application (Figure 5.1).

Layouter - implementation of layout algorithm. By default BioUML implements a set of layout algorithm.

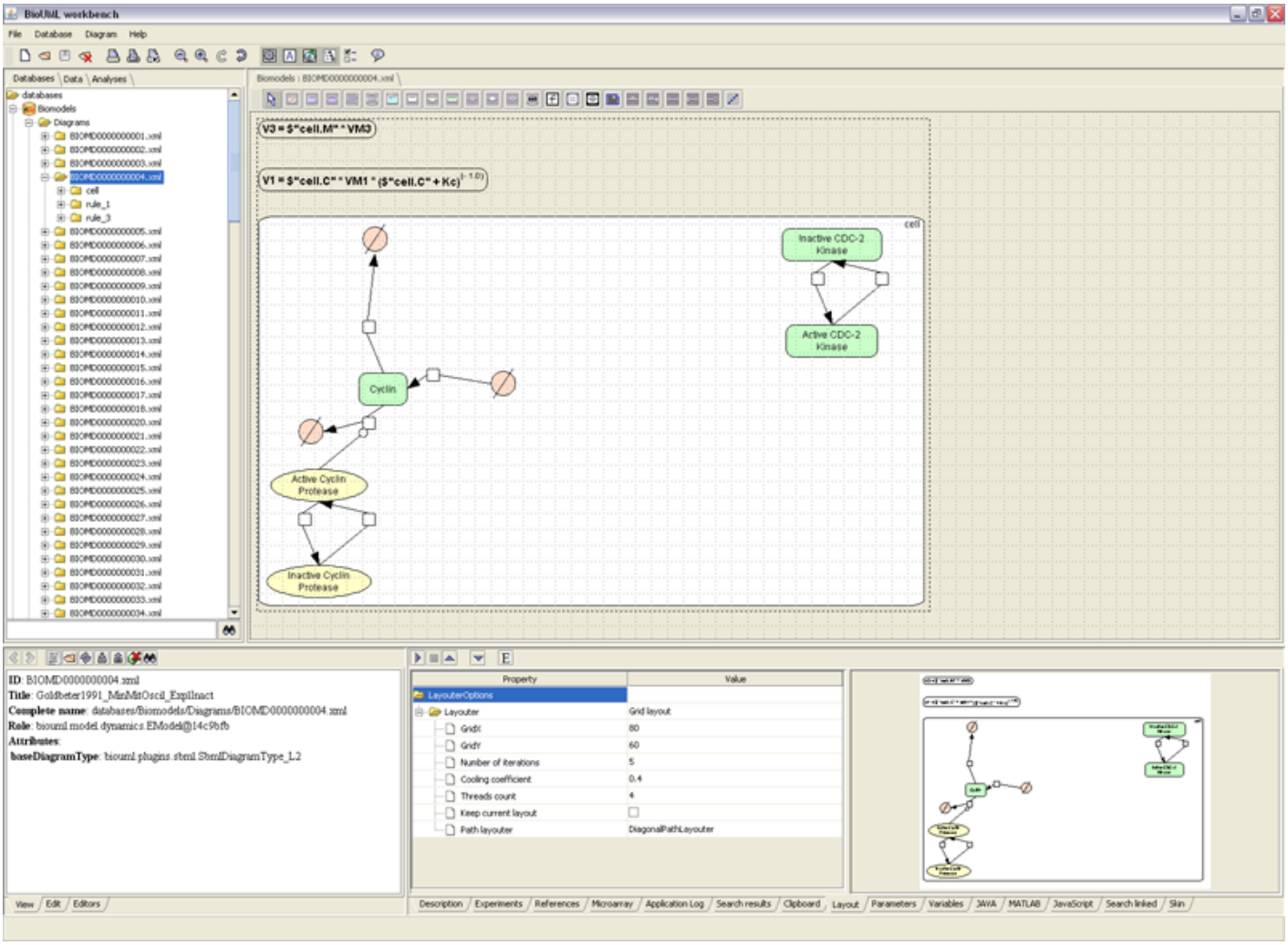


Figure 5.1. Diagram layout.

Layout toolbar actions:

- Prepare layout
- Stop layout
- Apply layout
- Save layout
- Expert mode

Typical usage of layout:

1. Select layouter and set preferences (for most case default preferences can be used).
2. Run **Prepare layout** action. If preview against your expectations run **Prepare layout** one more time with other preferences.
3. Run **Apply layout** action to apply the result to current opened diagram.

Layout algorithms:

- Cross cost grid layout
- Fast grid layout
- Hierarchic layout
- Force directed layout
- Orthogonal layout

Pathway layouter - internal instrument for layout decoration.

5.1 Prepare layout

Run layout process with selected preferences for current opened diagram. After complete layouted diagram image will be shown at the right of **Layout** view part and Apply layout action will be available

5.2 Stop layout

Stop current layout process. This action is usable when layout process is too long. After this action you can run new layout process with another parameters.

5.3 Apply layout

Apply the result of layout process to current opened diagram. All unmatched diagram elements will not be replaced.

5.4 Save layout

Move current diagram layout to **Layout view part**.

5.5 Expert mode

Go to expert mode of Layout preferences.

Expert mode allows to set all expert preferences of layout.

Example

Simple and expert preferences mode of Grid layouter

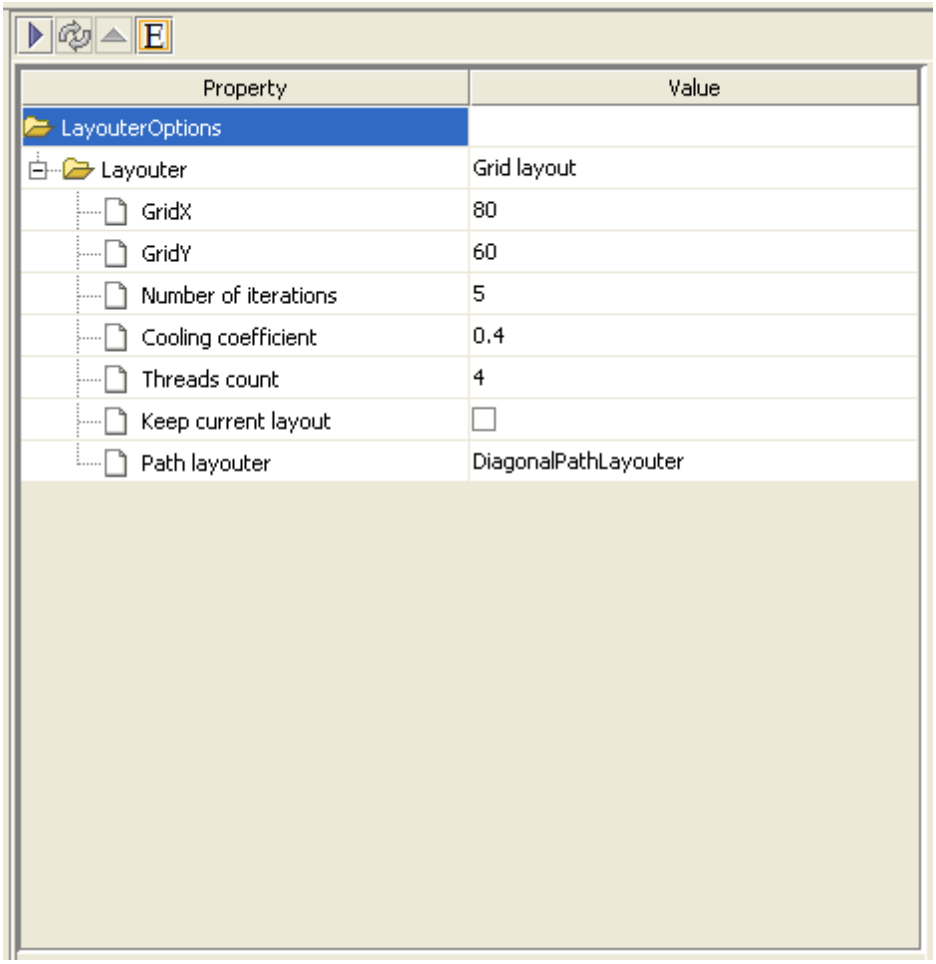


Figure 5.2. Simple preferences mode of Grid layout.

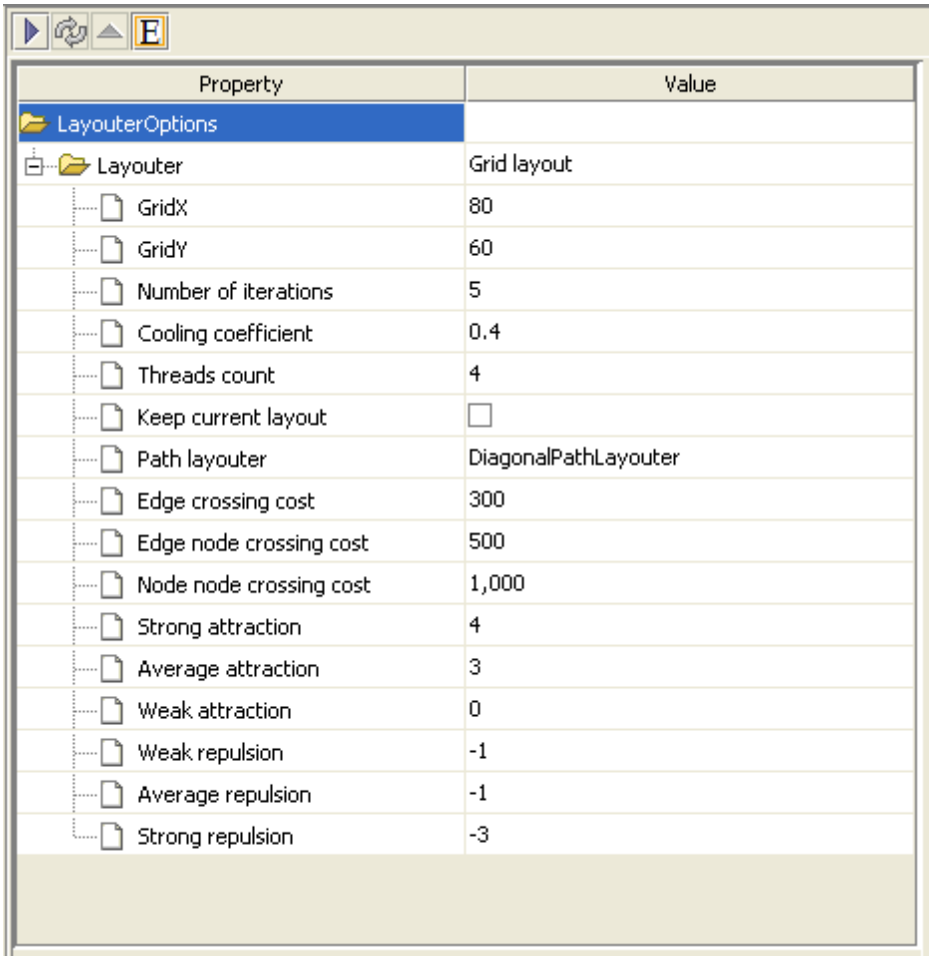


Figure 5.3. Expert preferences mode of Grid layout.

5.6 Graph layout algorithms

Properly drawn biological networks are of great help in the comprehension of their characteristics. In this topic we give a brief description of some graph layout algorithms. This stuff provide us with automatic visualization of different biological networks clears up their topological architectures and facilitate the understanding of the network functions.

Cross cost grid layout algorithm

Kato, M. *et al.*(2005) Automatic drawing of biological networks using cross cost and subcomponent data. *Genome Inform*., **16**, 22-31.

This is grid-based algorithm that considers (a) edge-edge crossings, (b) node-edge crossings, (c) node-node crossings, (d) distances between nodes in its cost function . This algorithm uses a weight matrix representing the difference between two sequentially obtained layouts for computing the costs, takes a greedy algorithm for searching locally optimal solutions and simulated annealing for global optimization.

The following parameters of the algorithm must be set:

- *GridX* - positive integer parameter represents horizontal grid step in pixels
- *GridY* - positive integer parameter represents vertical grid step in pixels
- *Number of iteration* - positive integer parameter represents number of iterations during same temperature in

simulated annealing

- *Cooling coefficient* - real number between 0 and 1 represents cooling coefficient of simulated annealing (small value decrease computation time but leads to quality loss)
- *Perturbation threshold* - real value parameter between 0 and 1 represents probability of layout perturbation during simulated annealing
- *Max distance* - positive integer parameter represents maximum repulsive distance expressed in number of grid steps (if distance between two nodes exceeds *Max distance* then there is no repulsion between them)

Fast grid layout algorithm

Kojima, K. *et al.* (2008) Fast grid layout algorithm for biological networks with sweep calculation. *Bioinformatics*., **24**, 1433-1441.

This algorithm has much in common with cross cost grid layout algorithm, described above, the main difference is in evaluating weight matrix. Fast grid layout algorithm applies for this purpose a method termed sweep calculation which reduce time complexity, making this algorithm faster rather then cross cost grid layout algorithm.

Algorithm uses similar to cross cost grid layout parameters. Some unique features are available:

- *Treads count* - positive integer parameter for parallel version of algorithm
- *Keep current layout* - if checked then algorithm starts with current layout, else starts with random layout
- *Path layouter* - parameter determines edge drawing style

Hierarchic layout algorithm

North S. C., Woodhull G. (1988) Online Hierarchical Graph Drawing. *Lecture Notes In Computer Science.,* **2265**, 232-246.

Algorithm works well on acyclic graphs and other graphs that can be drawn as hierarchies.

Force directed layout algorithm

One of many traditional realizations of force directed algorithms with repulsive forces between all nodes and attractive forces between nodes which are adjacent. Algorithm is fast but can't properly layout complicated graphs.

Orthogonal layout algorithm

This method uses greedy algorithm for nodes arrangement then draws orthogonal paths between them.

5.7 Pathway layouter

Pathway Layouter is internal feature, runs every time when some layout started. It is used as decorator for other graph layout algorithms to take into account some peculiarities of diagrams (for example all math elements are located on the top of diagram), besides it assumes layout of compartments if diagram contains them and selected algorithm does not support this feature.

6 JavaScript

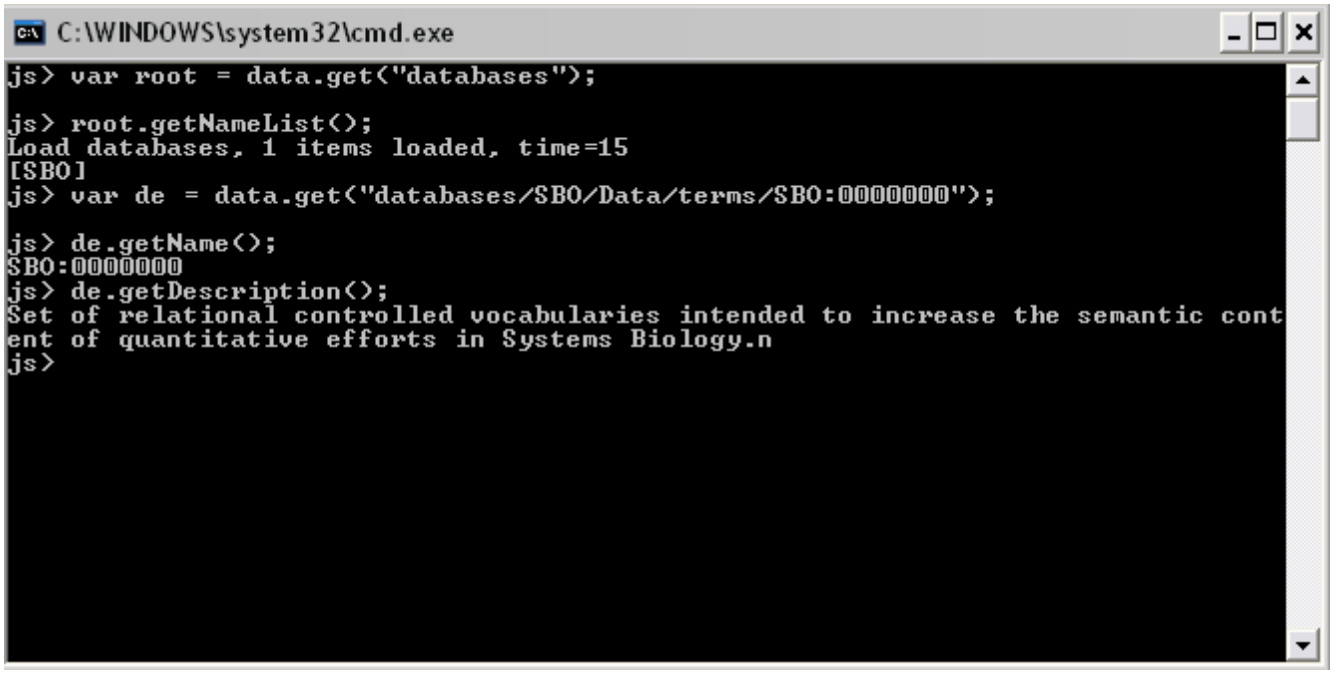
- Shell mode
- JavaScript tab
- JavaScript documents

6.1 Shell mode

BioUML workbench can be started in headless using JavaScript shell. This provides a simple way to run scripts in batch mode or an interactive environment for exploratory programming.

Use **shell.bat** to start BioUML JavaScript shell.

BioUML JavaScript shell provides a set of Custom functions and host objects. Below you can see an example of session where user gets element from database using **data** host object:



6.2 Custom functions and host objects

JavaScript plug-in defines two extension points that allows other plug-ins to expose their functions and objects to Shell mode , JavaScript tab or JavaScript documents.

ru.biosoft.plugins.javascript.function

This extension point allows plug-in to contribute its JavaScript functions. This JavaScript functions will be shown in **Analyses/JavaScript/Functions** section in repository tree. Function help will be shown in View/Edit tab when the function item will be selected in repository tree.

ru.biosoft.plugins.javascript.hostObject

Using this extension point plug-in can provide access to particular Java objects (host objects) provided by plug-in. Plug-in host oblects will be shown in **Analyses/JavaScript/Host** objects section in repository tree. Host object description (help) will be shown in View/Edit tab when the host object item will be selected in repository tree.

Standard host objects

- R - facade for R usage.
- data - facade for data-manipulations.
- dataFilter - facade for data-filtering.
- microarray - facade for microarray analysis.
- sbw - host object for SBW integration.

Documentation

Using described above extension points developer can provide description (documentation) for his functions and host objects.

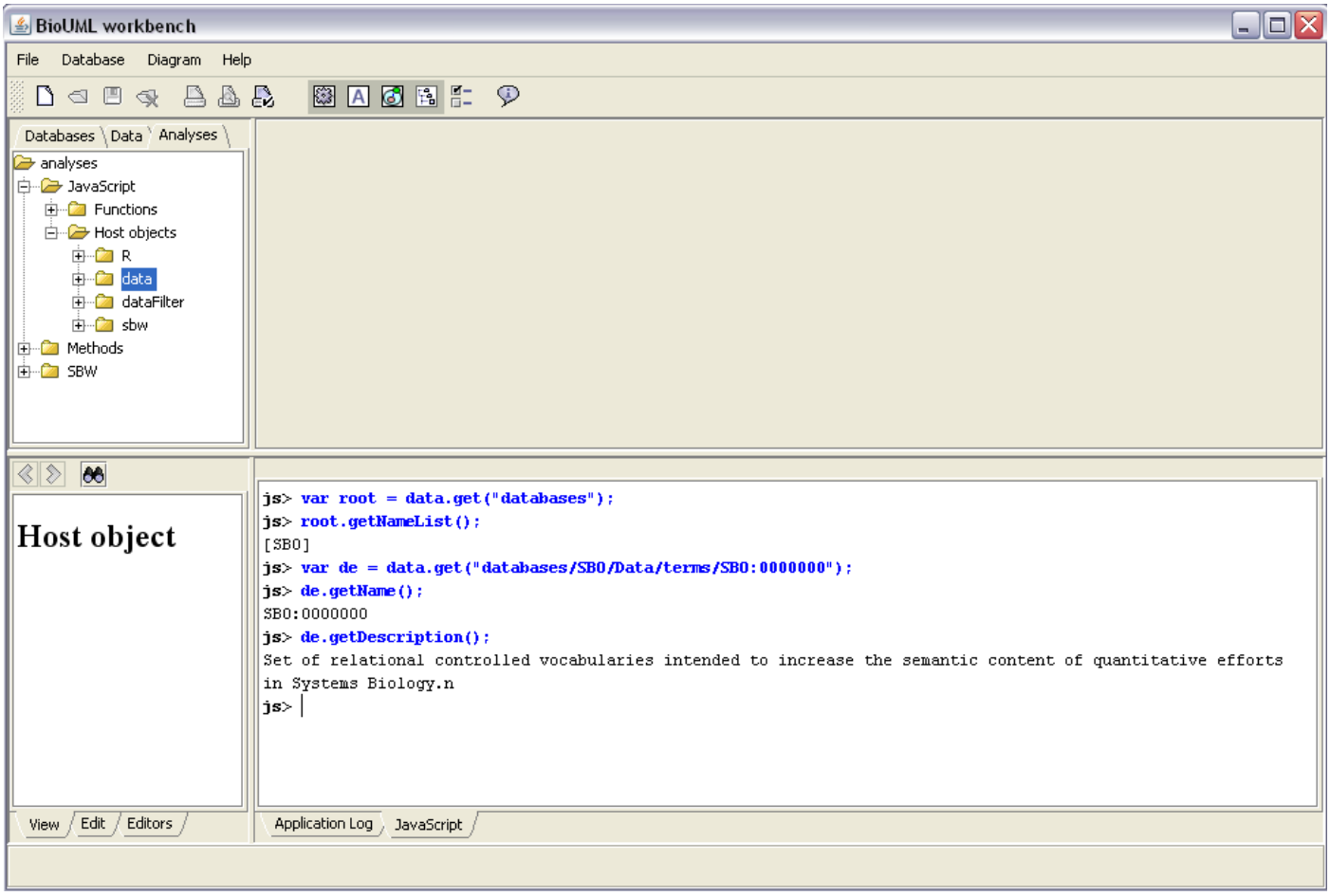
All functions and host objects for which help description is available will be shown in plug-in tree as children of JavaScript plug-in. When corresponding function or host object will be selected, its description will shown in Property Inspector pane.

Alternatively user can type function_name or help(function_name) and function description will be printed in JavaScript shell.

6.3 JavaScript tab

JavaScript tab is part of BioUML workbench, located in bottom left pane. It has the same functionality as Shell mode and provide command line for JavaScript inside BioUML workbench. Also it supports all Custom functions and host objects

Below you can see an example of JavaScript tab:



6.4 JavaScript documents

JavaScript document is a special type of data element which represents JavaScript file and allows user to edit, debug and execute scripts.

Default collection for JavaScript documents is **Data/scripts**. You can create new document using right click on **script** collection in repository and selecting **Add script**.

Note: the name of script should have suffix **js** (for example, **test.js**)

JavaScript documents supports Custom functions and host objects, so you can use it in the code.

Toolbar actions

There are special toolbar actions for JavaScript documents in BioUML:

- **Execute JavaScript** - start JavaScript execution until breakpoint or the end will be reached.
- **Execute selected part of document** - start execution of selected part only.
- **Execute one line** - execute next line of script. Use this action for debugging.

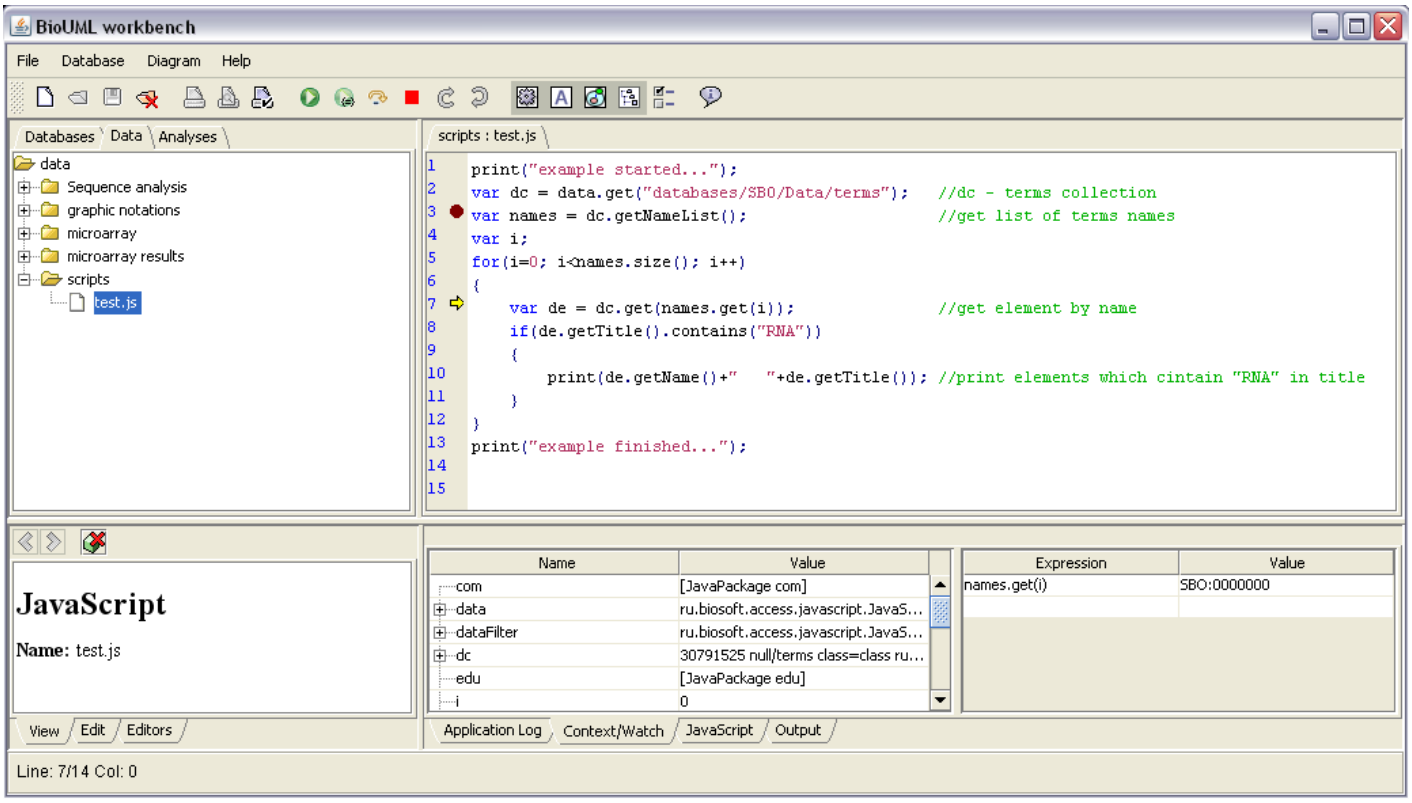
- **Break execution** - stop JavaScript execution

Debugging

You can set or remove breakpoint for line by clicking near line number.
To view variable value or expression value use **Context/Watch** tab

Example

Below you can see screenshot of example of using JavaScript documents in BioUML.



6.5 SBW JavaScript host object

SBW plug-in allows customer to explore available SBW modules, their services and methods.
SBW host object provides tight integration between SBW and BioUML JavaScript. Customer can write sophisticated scripts for analyses and simulation of SBML models.

sbw host object functions

- **help** – prints description of the specified SBW object.

  Example:

```
sbw.help("BROKER");
```

- **list** – prints list of all subelements for the specified SBW object, for example code below will print list of all methods for NOM service:

  Example:

```
sbw.list("edu.caltech.NOM/NOM");
```

- **getService** – returns service object for the specified SBW object. Getting service instance you can invoke any service method according to its SBW signature string.

  Example:

```
nom  =  sbw.getService("edu.caltech.NOM/NOM");
 descr  =  nom.getBuiltinFunctionInfo("usir")[0];
```

- **loadModel** – load SBML model with the given name from the specified BioUML module or file. To read SBML model as String for further analyses there is read method.

  Example of load model from the repository:

```
model  =  sbw.loadModel("SBML  model  reposytory",   // module name
                        "CellCycle-1991Gol.xml"); // model name
nom.loadModel(  model.read()  );
```

  Example of load of model from file:

```
model  =  sbw.loadModel("file",               // should be "file"
                        "c:/my_model.xml"); // model file name
```

7 R support

R is an integrated suite of software facilities for data manipulation, calculation and graphical display (**http://www.r-project.org**).

BioUML supports R script by **R** JavaScript host object.

R methods

- **local** - return RObject for locally installed R application using Java R Interface (JRI API). It's the fastest way for using R.

  Note: R_HOME environment variable should be set.

  Example:

```
var rObject = R.local();
```

- **rserve** - return RObject for locally installed R application using RServe service. This type of RObject is more stable and recommended to use with locally installed R application

  Note: If RServe is not running the system will try to run it automatically, otherwise you should run it manually.

  Example:

```
var rObject = R.rserve();
```

- **connect** - return RObject for remote R installation. In this case R should be installed on BioUML server. This method have two string parameters:

  host - address of BioUML server,

  port - connection port

  Example:

```
var rObject = R.connect("server.biouml.org", 80);
```

7.1 RObject methods

You can get RObject using R facade predefined object in JavaScript.

Each RObject object provides the set of methods:

- void **newSession**() - create new R session, all variables in previous context will be removed.

  Example:

```
r.newSession();
```

- void **saveSession**(String name) - associate current context with name and save it on file system. Context can be

restored later with **openSession** function

Example:

```
r.saveSession("mySession");
```

- void **openSession**(String name) - remove current R variables and load context from file system by name.

Example:

```
r.openSession("mySession");
```

- void **voidEval**(String expression) - evaluate R expression in R environment without result returning.

Example:

```
r.voidEval("a <- 10"); //create variable in R context with integer valu
```

- REXP **eval**(String expression) - evaluate R expression in R environment. The parameter of this method is string with R script and it returns object as a result of the last R commend.

Example:

```
var d = rObject.eval("rnorm(2)"); //return array of double
```

- void **assignObject**(String varName, Object value) - assign Javascript object to R environment with specified variable name.

Example:

```
rObject.assignObject("a", [1,2,3]); //add array variable "a" to R envi
```

- void **help**(String expression) - looks for help in R environment and display it in BioUML environment.

Example:

```
rObject.help("data"); //display Data Sets help page from R
```

Complete example

Here is a complete JavaScript example of R session in BioUML.

```
var rObject = R.connect("server.biouml.org", 80);    //get remote RObject
print(rObject);                                       //print rObject to che
rObject.newSession();                                 //clean R context
rObject.voidEval("x=0;for(i in 1:100) {x = x+i;});   //execute set of commar
var x = rObject.eval("x");                            //get x value from R
```

```
print(x);                                                    //print the result to c
rObject.saveSession("testSession");                          //save current variable
```

To use x variable value next time you can use script like this:

```
var rObject = R.connect("server.biouml.org", 80);    //get remote RObject
rObject.openSession("testSession");                  //load saved R session
var x = rObject.eval("x");                           //get variable from R
print(x);                                            //print x value to cons
```

7.2 R graphic support

Graphical facilities are an important and extremely versatile component of the R environment. BioUML supports R graphic component with rplot JavaScript function:

rplot(RObject r, String expression)

r - RObject object;

expression - R expression which contains R graphic commands

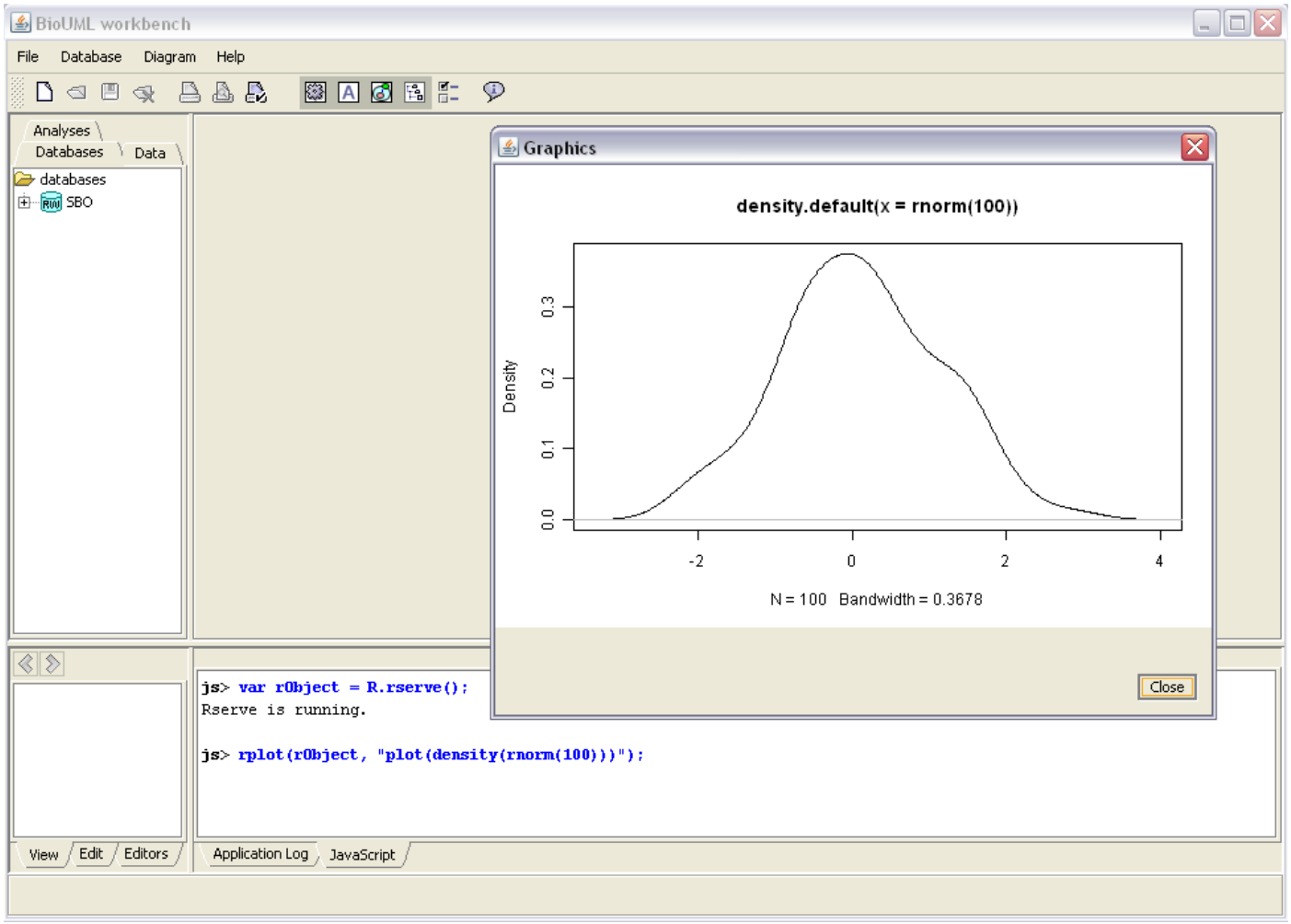
The result of rplot function is Graphic Dialog in workbench version of BioUML and new browser window with picture in Web version.

Note: prlot will not display a result in console version of BioUML.

Simple example

Draw simple graphic with random values

```
var rObject = R.rserve();                          //get RObject
rplot(rObject, "plot(density(rnorm(100)))");     //execute graphic script
```

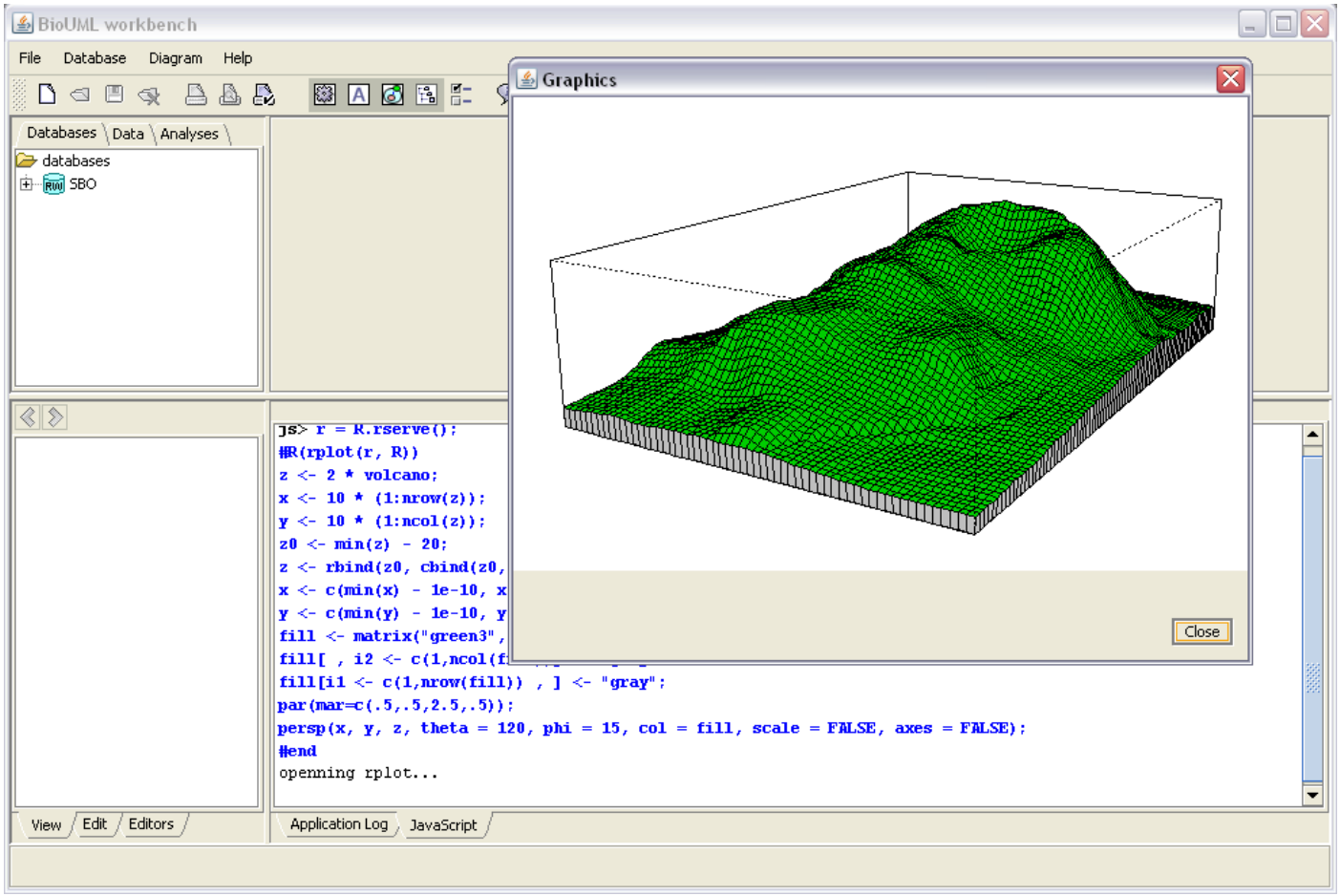


Complex example

Draw 3D graphic using R preprocessor

```
r = R.rserve();                                          //get RObject
#R(rplot(r, R))                                          //start block of commands
 z <- 2 * volcano;
 x <- 10 * (1:nrow(z));
 y <- 10 * (1:ncol(z));
 z0 <- min(z) - 20;
 z <- rbind(z0, cbind(z0, z, z0), z0);
 x <- c(min(x) - 1e-10, x, max(x) + 1e-10);
 y <- c(min(y) - 1e-10, y, max(y) + 1e-10);
 fill <- matrix("green3", nr = nrow(z)-1, nc = ncol(z)-1);
 fill[ , i2 <- c(1,ncol(fill))] <- "gray";
 fill[i1 <- c(1,nrow(fill)) , ] <- "gray";
```

```
 par(mar=c(.5,.5,2.5,.5));
persp(x, y, z, theta = 120, phi = 15, col = fill, scale = FALSE, axes = FA
#end                                                    //end block of commands
```



7.3 R preprocessor

R preprocessor allows you to use R language directly instead of parameter of JavaScript functions.
R script should be located in special block inside JavaScript which begins from **#R(...)** and ends with **#end**

In next table possible variants presented with equivalent JavaScript code:

With preprocessor

```
r = R.rserve();
#R(r)
    c = rnorm(50);
#end
```

Without preprocessor

```
r = R.rserve();
r.eval("c = rnorm(50)");
```

```
r = R.rserve();
#R(r.voidEval(R))
    c = rnorm(50);
#end
```

```
r = R.rserve();
r.voidEval("c = rnorm(50)");
```

```
r = R.rserve();
#R(rplot(r,R))
    c = rnorm(50);
    hist(c);
#end
```

```
r = R.rserve();
rplot(r,"c = rnorm(50);hist(c);");
```

7.4 Using R script directly

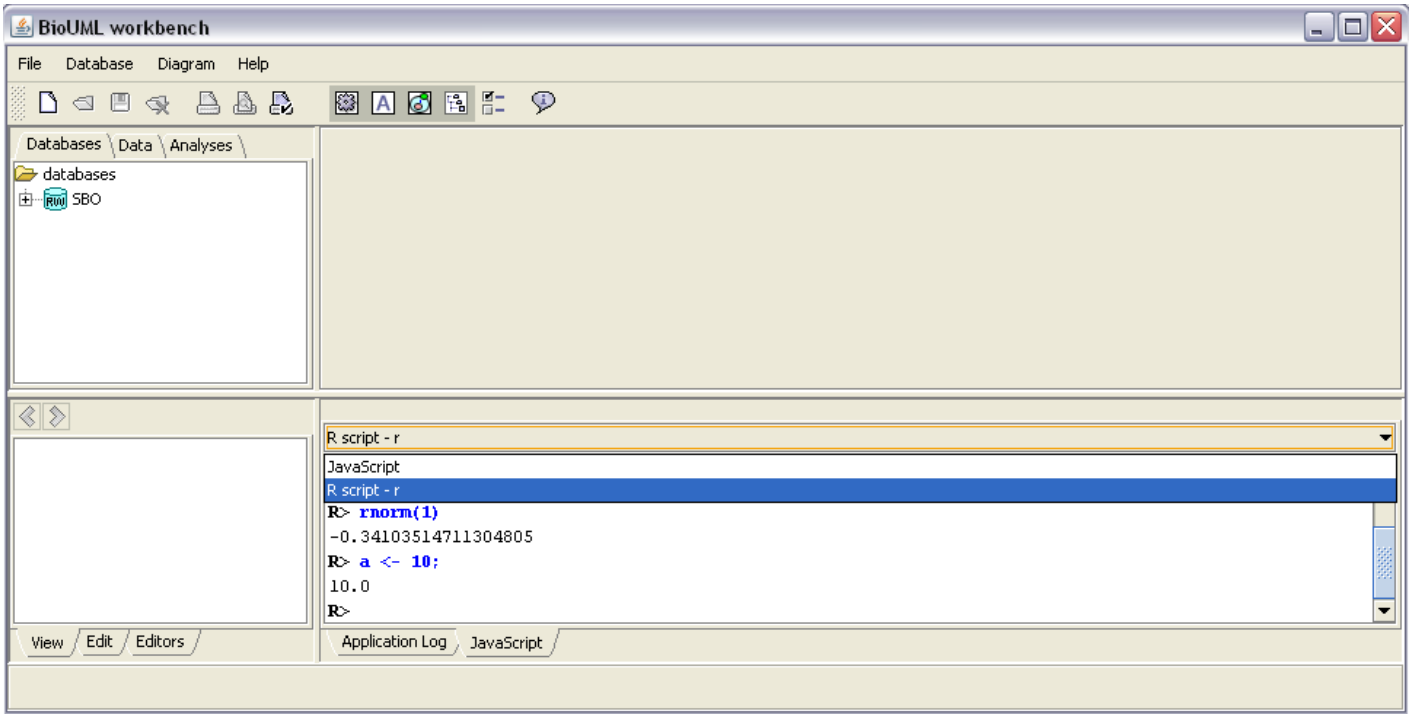
Except RObject functions and R preprocessor in BioUML workbench user can type R commands directly in R syntax.

JavaScript pane provides combo box with available types of preprocessors. This preprocessor will be used for all commands in text area. Default preprocessor JavaScript doesn't makes any preprocessor operations.

To add R preprocessor you should only create RObject. For example,

```
r  = R.rserve()
```

After this command "**R script - r**" item will be added to preprocessor list



With **R script - r** all commands will be automatically transform from R to JavaScript with **r** object.

8 Graphic notation editor

Diagram type (graphic notation) can be defined formally as XML document. Graphic notation editor provides user friendly interface to edit diagram types (Figure 5.1).

Several graphic notations were developed using graphic notation editor:

- SBGN – Systems Biology Graphic Notation (Le Novère et al, 2009)
- KEGG notation – shows metabolic pathways like KEGG meatabolic maps.

See also

- Diagram types
- Formal definition of graphic notation
- http://www.biouml.net/demo/Graphic_Notations_Editor.exe - flash movie about Graphic Notation Editor

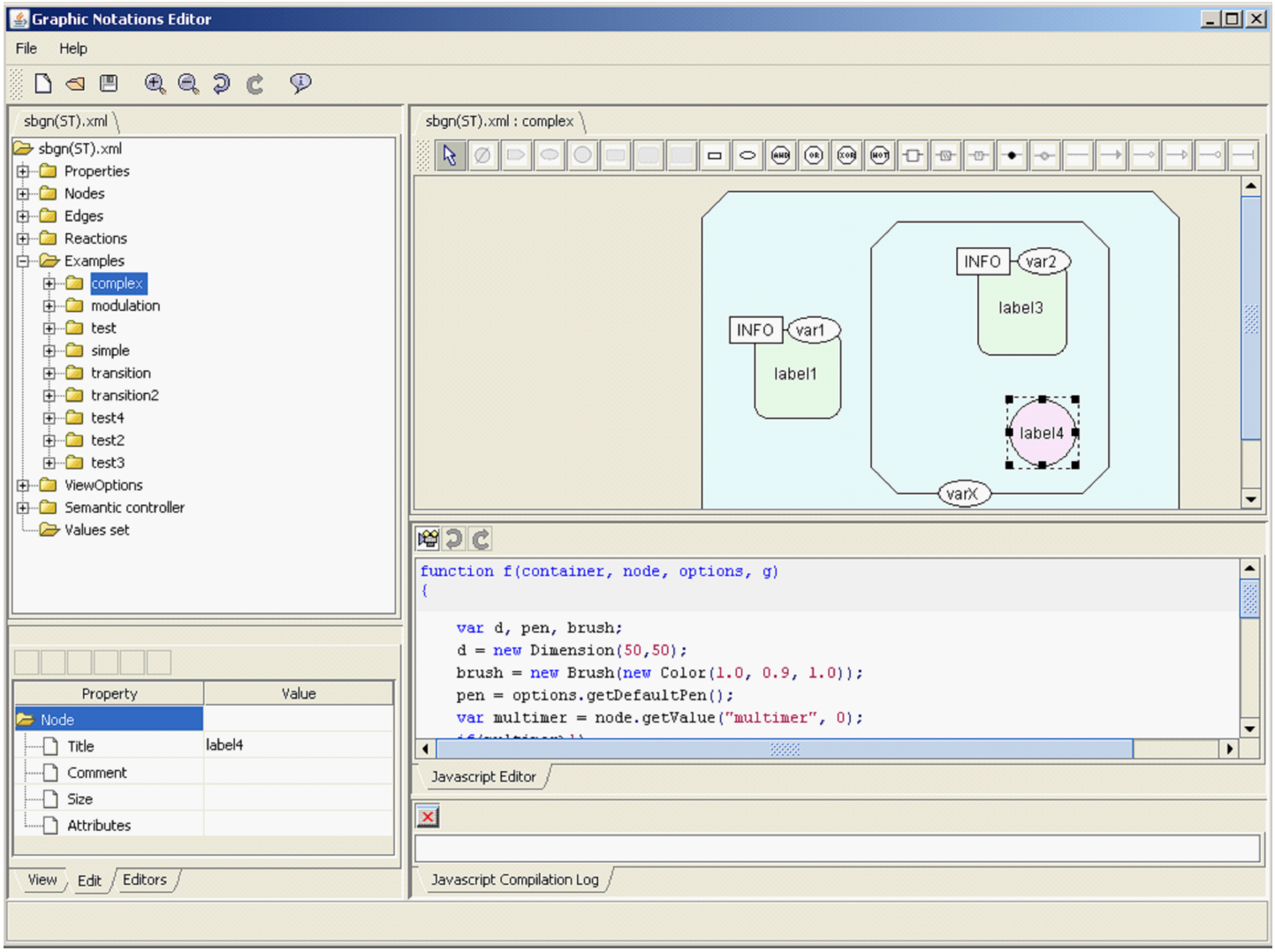


Figure 5.1. Graphic notation editor. Top left pane – graphic notation tree, bottom left pane – properties of selected object, top right pane – diagram, middle right pane – JavaScript code for drawing selected node type; bottom right

pane – Javascript log.

8.1 Formal definition of graphic notation

Graphic notation can be defined formally as XML document.

Main XML tags and properties are summarized in Table 1. It should be noted that XML document includes JavaScript function to generate images for nodes and edges.

See also

- http://www.biouml.org/sbgn.shtml - more detailed description of graphic notation concept, XML format and implementation details
- http://www.biouml.org/_sbgn/graphic%20notation.dtd - DTD for XML document that defines graphic notation
- http://www.biouml.org/_sbgn/SBGN%20-%20formal%20definition.doc – description of main concepts and SBGN implementation

Table 1.

Formal definition of graphic notation as XML document

Graphic notation components (XML tag)	Main properties (XML attributes)	JavaScript	Comment
properties property	name type short description controlled vocabulary	-	formal definition of properties that can be used as properties of nodes and edges (for example, title, multimer, etc.)
nodes node	name icon properties short description	function to generate node view	definition of node types
edges edge	name icon properties short description	function to generate edge view	definition of edge types
reactions reactions	name icon properties short description	function to generate reaction view	definition of reaction types
semanticController		○ canAccept ○ isResizable	defines rules for semantic control of diagram

Graphic notation components (XML tag)	Main properties (XML attributes)	JavaScript	Comment
		○ move	integrity. For this purpose it defines corresponding JavaScript functions.
viewOptions property	name type short description		view options to generate node or edge view
examples dml			a set of diagrams that can be used as test cases, legend and examples for the graphic notation. DML - Diagram Markup Language – is used for this purpose.

8.1.1 SBGN example - simple chemical

A simple chemical is defined by opposition to the macromolecule, as a chemical compound that is not formed by the covalent linking of pseudo-identical residues. Examples of simple chemicals are: an atom, a monoatomic ion, a salt, a radical, a solid metal, a crystal etc.

Specification

container:

A simple chemical is represented by a circular container.

label:

The identification of the simple chemical is carried by an unbordered box containing a string of characters. The characters may be distributed on several lines to improve readability, although this is not mandatory. The label box has to be attached to the center of the container box. Note that the label can spill-over from the container box.

auxiliary items:

A simple chemical can carry one or several units of information. A particular unit of information carries the physical type of the simple chemical.

name label SBO term
non-macromolecular ion mt:ion SBO:0000327
non-macromolecular radical mt:rad SBO:0000328

SBGN specification

BioUML implementation

Formal definition

```
<node icon="icon3" type="simple chemical">
<propertyRef name="multimer" value="0"/>
</node>

<nodeView type="simple chemical"><![CDATA[function f(container, node, options, g)
{
    var d, pen, brush;
    d = new Dimension(50,50);
    brush = new Brush(new Color(1.0, 0.9, 1.0));
    pen = options.getDefaultPen();
    var multimer = node.getValue("multimer", 0);
    if(multimer>1)
    {
          var ellipse = new EllipseView(pen, brush, 5, 5, d.width, d.height);
          container.add(ellipse);
    }
    var ellipse = new EllipseView(pen, brush, 0, 0, d.width, d.height);
    container.add(ellipse);
    var title = new TextView(node.getValue('title', ''), options.getNodeTitleFont(), g);
    container.add(title, CompositeView.X_CC | CompositeView.Y_CC);
    if(multimer>1)
    {
          var mbox = new BoxView(pen, brush, d.width/2-10, -10, 20, 20);
          container.add(mbox);
          var mcount = new TextView(node.getValue('multimer', '...'), options.
getNodeTitleFont(), g);
          var mPosition = new Point(d.width/2-5, 2);
          container.add(mcount, CompositeView.X_LL | CompositeView.Y_TT, mPosition);
    }
    return false;
}]]></nodeView>
```

8.1.2 SBGN example - macromolecule

Biological processes involve many different macromolecules that are built upon the covalent linking of pseudo-identical units. Examples are proteins, nucleic acids (RNA, DNA) or polysaccharides (glycogen, cellulose, starch . . .). In order to limit the explosion of symbols to be absorbed by the community, SBGN Level 1 defines only one glyph that represents a macromolecule. The same glyph is to be used for a protein, a nucleic acid, a complex sugar etc. Further Levels of SBGN might subclass this concept and propose different glyphs for different types of macromolecules.

Specification

container:

A macromolecule is represented by a round-corner rectangular container.

label:

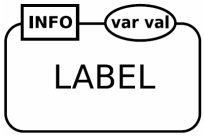
The identification of the macromolecule is carried by an unbordered box containing a string of characters. The characters can be distributed on several lines to improve readability, although this is not mandatory. The label box has to be attached to the center of the container box. Note that the label can spill-over from the container box.

auxiliary items:

A macromolecule can carry a certain number of state variables, that will add information about its precise state. The state of a macromolecule is therefore defined as the vector of all the state variables. A state variable is

represented by an ellipsoid. The large axis of the ellipsoid is located on the border of the macromolecule container box. The label of the state variable (type of characteristic, residue type, residue number) can be optionally written, either within the macromolecule container box, beside the border of the state variable box, or within the modificator box itself.

A macromolecule can also carry one or several units of information. Those units of information can characterise a domain, such as a binding site, or an exon. Particular units of information carry the material type of the macromolecules and the conceptual types of the macromolecules. The center of the bounding box of a unit of information is located on the midline of the border of the compartment.



SBGN specification

BioUML realization

Formal definition

```
<node icon="icon4" isCompartment="true" type="macromolecule">
<propertyRef name="multimer" value="0"/>
</node>

<nodeView type="macromolecule"><![CDATA[function f(container, node, options, g)
{
    var d, pen, brush;
    d = node.getValue("shapeSize", null);
    if(d.width == 0 && d.height == 0)
    {
        d.width = 70;
        d.height = 40;
    }
    brush = new Brush(new Color(0.9, 1.0, 0.9));
    pen = options.getDefaultPen();
    var multimer = node.getValue("multimer", 0);
    if(multimer>1)
    {
         var rect = new RoundRectangle2D.Float(5, 5, d.width, d.height, 20, 20);
         var box = new BoxView(pen, brush, rect);
         container.add(box);
    }
    var rect = new RoundRectangle2D.Float(0, 0, d.width, d.height, 20, 20);
    var box = new BoxView(pen, brush, rect);
    container.add(box);
    var title = new TextView(node.getValue('title', ''), options.getNodeTitleFont(), g);
    container.add(title, CompositeView.X_CC | CompositeView.Y_CC);
    if(multimer>1)
    {
         var mbox = new BoxView(pen, brush, 10, -10, 20, 20);
         container.add(mbox);
         var mcount = new TextView(node.getValue('multimer', '...'), options.
getNodeTitleFont(), g);
         var mPosition = new Point(15, 2);
         container.add(mcount, CompositeView.X_LL | CompositeView.Y_TT, mPosition);
```

```
    }
    return false;
}]]></nodeView>
```

9 Analysis Methods

9.1 Optimization

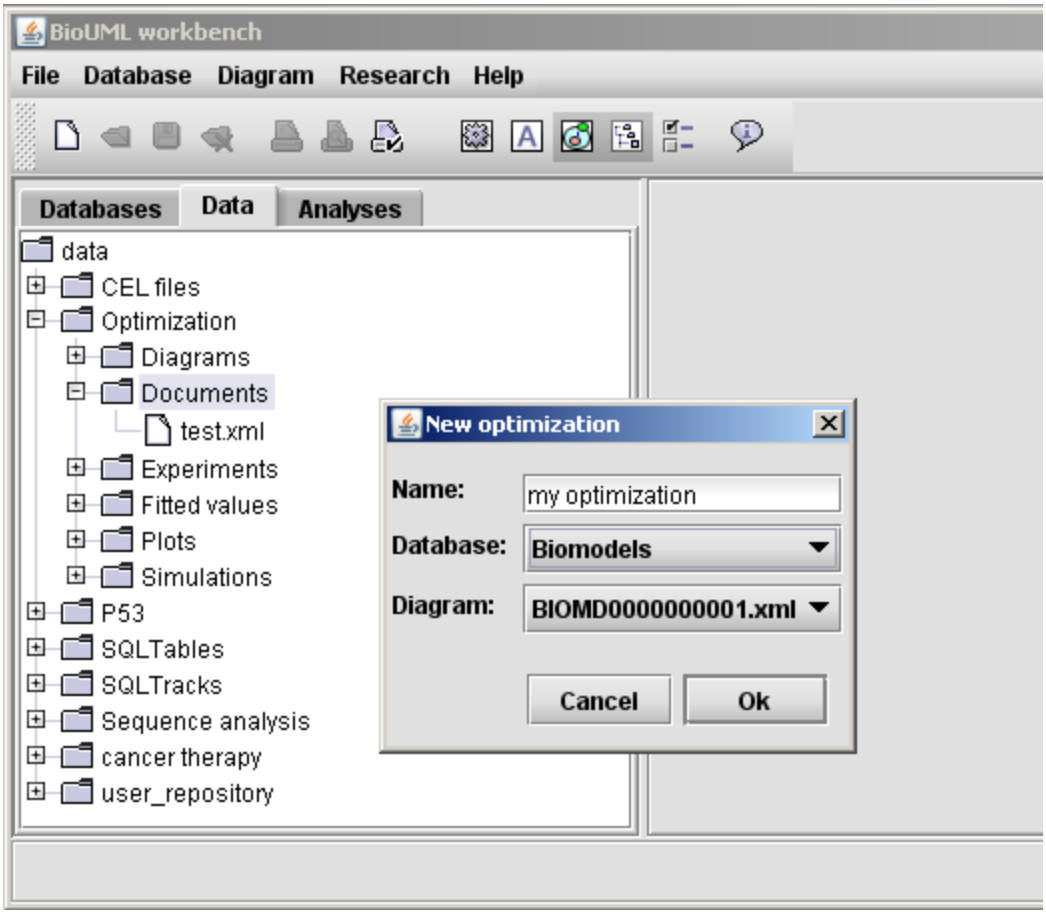
BioUML has the opportunity to estimate the diagram parameters for minimizing a deviation of the diagram simulation results from the experimental datasets. These datasets represent the result of time courses expressed as exact or relative values of substance concentrations and can be found in the literature or generated by BioUML. The optimization plug-in supports some different methods attempting to minimize the objective function, which is the sum of the squares of the distances between the simulation results and the experimental dots, taking into account the mean-square weighted criterion.

Implement the following steps to start optimization.

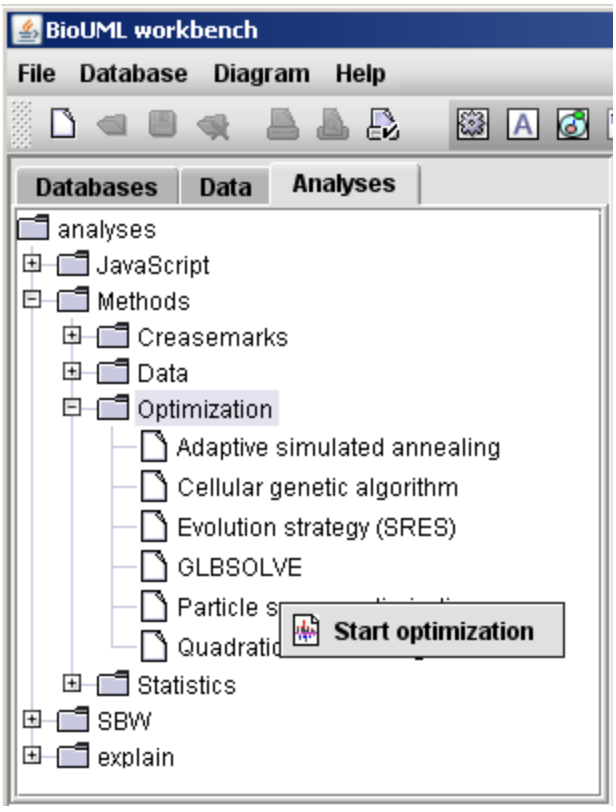
- Create the optimization document.
- Select the optimization method and specified its parameters.
- Indicate parameters to fit.
- Create additional parameter constraints if necessary.
- Specified experimental data.
- Indicate one of the available solvers in "Simulation" tab of the optimization document.
- Click on the "Start" button of the document "Method" tab.

9.1.1 Optimization document

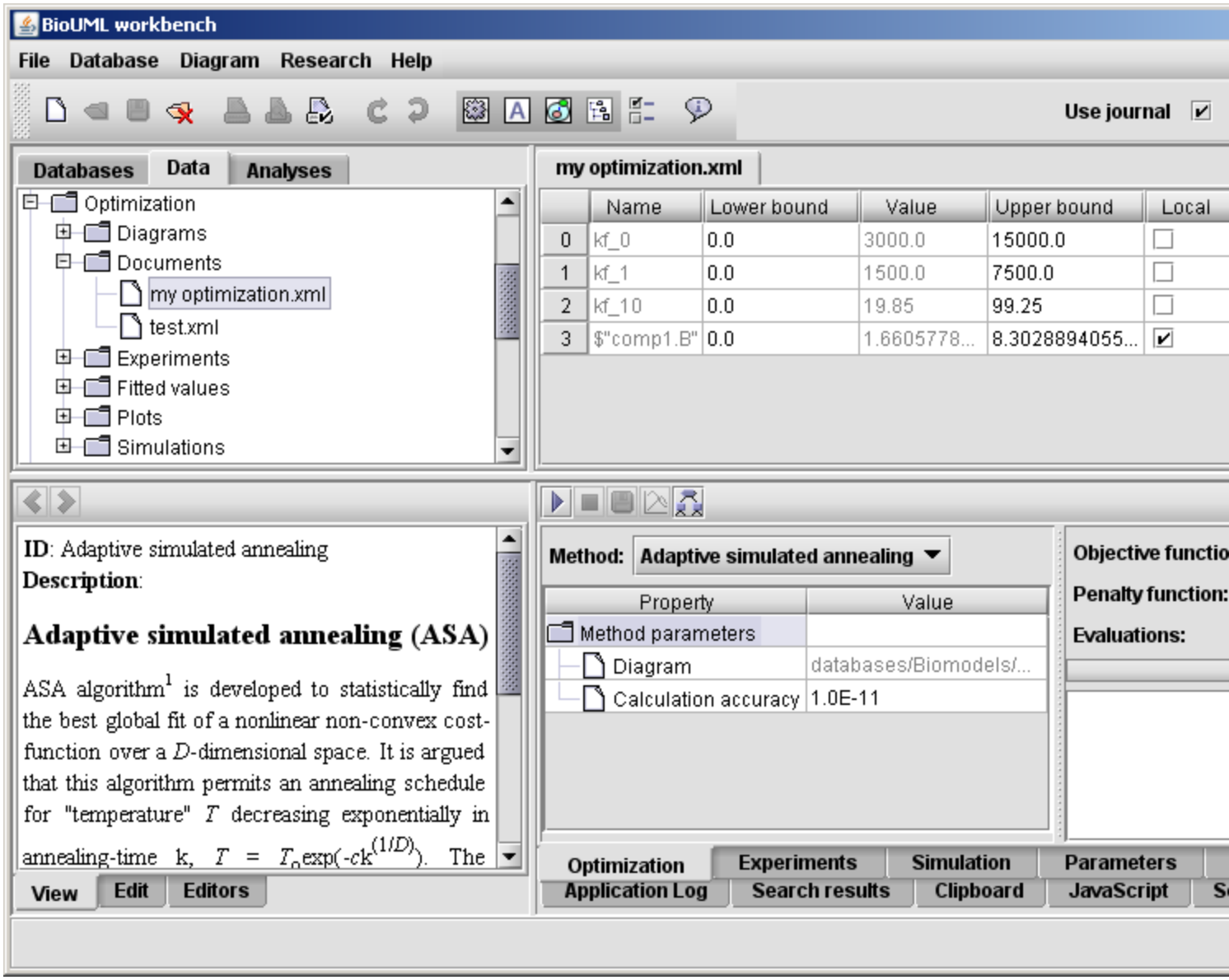
BioUML allows performing diagram parameters estimation by creation of a special optimization document. This document is stored as xml-file and combines all the settings required for the optimization including a list of parameters to fit indicating the search boundaries, parameters of a selected optimization method, solver settings and the experimental data list. When optimization is complete, the resulting parameter values and results of model simulations can be stored as tables and then plotted. To create a new document select tab "Data" at the top pane of the dialog and open folder "Optimization". Then click the left mouse button on "documents" folder and choose item "New optimization" in the pop-up menu. Type the name of the document in the resulting input dialog, select the database and the diagram which you want to operate and press "OK". The existing document can be opened by the double-click. If you want to remove a document, click the left mouse button on it and select item "Remove".



Another way to create an optimization document is as follows. Select tab "Analysis" to the left of tab "Data". Find the optimization methods list in the tree under "Methods" -> "Optimization". Click the left mouse button on the method by which you want to perform optimization and select "Start optimization" in the pop-up menu.



The working area of the optimization document includes the three main windows. The first window located in the lower right corner of the framework is designed to select the optimization options. It includes tabs with optimization method and diagram parameters, experimental data and solver settings. "Method" tab of this window, in particular, contains the buttons to start and stop optimization, to save the results and to draw the plots for their visual presentation. It also informs you about the calculation progress and the best (smallest) values found for the objective and penalty functions up to this time. To the left of this window you can see a short description of the chosen optimization method and links to the base papers for this method programming. Finally the upper right window is intended to store parameters for fitting.



9.1.2 Optimization methods

The optimization methods described in this chapter attempt to minimize the objective function which is the sum of the squares of the distances between the diagram simulation results and the experimental dots. This function is defined as follows

$$J = ?_{i,j} w_j (y_{\text{pred}}(i,j) - y_{\text{exp}}(i,j))^2,$$

where the indexes i and j denote rows and columns in the dataset, y_{exp} represents the known experimental data and y_{pred} is the corresponding simulated values. The weight w_j calculated for each data column by one of the formula $w_j = 1/|\langle x_j \rangle|$, $w_j = (\langle x_j^2 \rangle)^{-1/2}$ or $w_j = 1/|\langle x_j^2 \rangle - \langle x_j \rangle \langle x_j \rangle|$, where $\langle x_j \rangle$ is the mean value of the points in a trajectory of experimental dots. Users also are able to individually override these weights.

Minimization of the objective function can be performing by one of the following optimization methods.

- Adaptive simulated annealing (ASA)

- Stochastic ranking evolution strategy (SRES)
- Global optimization using the DIRECT algorithm (GLBSOLVE)
- Quadratic Hill-climbing
- Multi-objective cellular genetic algorithm (MOCell)
- Multi-objective particle swarm optimization (MOPSO)

Comparative characteristics of the methods are given by the following table.

Method	References	Year	Parallel	Parameter constraints
ASA	[1]	1996	-	+
SRES	[2]	2000	+	+
GLBSOLVE	[3], [4]	1999	-	-
Hill-climbing	[5]	1965	-	+
MOCell	[6]	2005	+	+
MOPSO	[7], [8]	1995	+	+

Below is a brief description of the methods.

Adaptive simulated annealing (ASA)

Adaptive simulated annealing (ASA) is a global optimization algorithm that relies on randomly importance-sampling the parameter space [1]. ASA algorithm is developed to statistically find the best global fit of a nonlinear non-convex cost-function over a D-dimensional space. It is argued that this algorithm permits an annealing schedule for "temperature" T decreasing exponentially in annealing-time k,

$$T = T_0 \exp(-ck^{(1/D)}).$$

The introduction of re-annealing also permits adaptation to changing sensitivities in the multi-dimensional parameter-space. This annealing schedule is faster than fast Cauchy annealing, where $T = T_0/k$, and much faster than Boltzmann annealing, where $T = T_0/\ln k$.

The following options of the algorithm must be set.

- *Calculation accuracy* is a positive double value to determine the difference between deviation of simulation result from experimental dataset under parameter values of current calculation step and values of previous calculation step. The default is '1.0E-11'.

Stochastic ranking evolution strategy (SRES)

The Evolution Strategy using Stochastic Ranking [2] is a $(\mu\ \lambda)$-ES evolutionary optimization algorithm that uses stochastic ranking as the constraint handling technique. In the $(\mu\ \lambda)$-ES algorithm, the individual i is a set of real-valued vectors $(x_i,\ \sigma_i)$ for all $i = 1,...,\lambda$. The initial population of x is generated according to a uniform n-dimensional probability distribution over the search space S. Let δx be an approximate measure of the expected distance to the

global optimum, then the initial setting for the "mean step sizes" should be

$$\Gamma_{i,j}^{(0)} = \delta x_j / n^{1/2} \sim (l_j - r_j) / n^{1/2},\ l_j = x_j = r_j,\ i = 1,...,\lambda;\ j = 1,...,n,$$

where $\sigma_{i,j}$ denotes the jth component of the vector σ_i. We use these initial values as upper bounds on σ. Following the bubble-sort-like procedure is used to rank the individuals in a population, and the best (highest ranked) μ individuals out of λ are selected for the next generation. The truncation level is set at $\mu / \lambda \sim 1/7$. Variation of strategy parameters is performed before the modification of objective variables. We generate λ new strategy parameters from μ old ones so that we can use the λ new strategy parameters in generating λ offspring later. The "mean step sizes" are updated according to the log-normal update rule: $i = 1,...,\mu$; $h = 1,...,\lambda$; $j = 1,...,n$,

$$\Gamma_{h,j}^{(g+1)} = 1/2\ (\sigma_{h,j}^{(g)} + \sigma_{k,j}^{(g)})\ \exp(\tau_2 N(0,1) + \tau\ N_j(0,1)) \quad \textbf{(1)}$$

where $N(0,1)$ is a normally distributed one-dimensional random variable with an expectation of 0 and variance 1 and $k = 1,...,\mu$ is an index generated at random and anew for each j. The "learning rates" τ and τ_2 are set equal to $(4n)^{-1/4}$ and $(2n)^{-1/2}$, respectively. Recombination is performed on the self-adaptive parameters before applying the update rule given by (1). Having varied the strategy parameters, each individual $(x_i,\ \sigma_i)$, for all $i = 1,...,\mu$ creates λ/μ offspring on average, so that a total of λ offspring are generated

$$x_{h,j}^{(g+1)} = x_{h,j}^{(g)} + \sigma_{h,j}^{(g+1)}\ N_j(0,1).$$

Recombination is not used in the variation of objective variables. When an offspring is generated outside the parametric bounds defined by the problem, the mutation (variation) of the objective variable will be retried until the variable is within its bounds. In order to save computation time, the mutation is retried only ten times and then ignored, learning the object variable in its original state within the parameter bounds.

The following options of the algorithm must be set.

- *Number of iterations* is a positive integer value to determine the number of iterations the algorithm shall evolve the population. The default is '1750'.
- *Population size* is a positive integer value to determine the number of individuals that survive after each iteration. The default is '20'.

Global optimization using the DIRECT algorithm (GLBSOLVE)

A deterministic GO method [3] is a version of the DIRECT algorithm [4]. The first step in the DIRECT algorithm is to transform the search space to be the unit hypercube. The function is then sampled at the center-point of this cube. Computing the function value at he center-point instead of doing it at the vertices is an advantage when dealing with problems in higher dimensions. The hypercube is then divided into smaller hyperrectangles whose center-points are also sampled. Instead of using a Lipschitz constant when determining the rectangles o sample next, DIRECT identifies a set of potentially optimal rectangles in each iteration. All potentially optimal rectangles are further divided into smaller rectangles whose center-points are sampled. When no Lipschitz constant is used, there is no natural way of defining convergence (except when the optimal function value is known as in the test problems). Instead, the procedure described above is performed for a predefined number of iterations. Note that this solver does not support parameters constraints.

The following options of the algorithm must be set.

- *Number of iterations* is a positive integer value to determine the number of iterations the algorithm shall

divide the hypercube (search space) into smaller hyperrectangles. The default is '1500'.

Quadratic Hill-climbing

The gradient method [5] for minimazing general function $-H(x)$, where variable x denotes the column vector of variables $(x_1,...,x_n)$. This method rests on maximizing a quadratic approximation to the function $H(x)$ on a suitably chosen spherical region. Computational technique for maximization take the form of an iterative procedure. The method is specifically designed to work for functions which are not concave and for starting points which are not necessarily near a maximum. Let us denote by F_x the vector of first partial derivatives evaluated at x and by S_x the symmetric

(n x n) matrix of second partial derivatives evaluated at x. Assume that $H(x)$ admits of a second-order Taylor series expansion around a point

$a = (a_1,...,a_n)$:

$$H(x) \tilde{} H(a) + (x - a)'F_a + \tfrac{1}{2}(x - a)'S_a(x - a) \quad \textbf{(1)}$$

where the subscripts indicate the point of evaluation. The iterative procedure for finding the maximum of the function is, given point x_p at which S_x^p and F_x^p are evaluated, to define the next point, x_{p+1}, as the maximum of the quadratic approximation (1) on a spherical region centered at x_p. Ideally, the region should be taken as large as possible provided that it is small enough that in the region the quadratic approximation is a satisfactory guide to the actual behavior of the function. The following procedure attempts to approximate this ideal. Two distinct cases arise:

(a) F_x^p significantly different from 0. In this event we choose a number

$$\pm = \lambda_1 + R\|F_x^p\| \quad \textbf{(2)}$$

where λ_1 is the largest eigenvalue of S_x^p, and R is a positive parameter determined by the following rule. Let ΔH be the actual change in the function due to the proposed Δx and let ΔQ be the corresponding change in the quadratic approximation. Let $z = \Delta H/\Delta Q$. If $z = 0$, the proposed Δx implies overshooting; it is therefore not accepted, R is increased by a factor of 4 and a new $(S - \alpha I)^{-1}$ is calculated. If $z > 0$ and close to unity (in practice, if z is between 0.7 and 1.3) R is decreased by multiplying R by a factor of 0.4. If $z > 2$, R is again increased by a factor of 4. For other values of z ($0 = z_d$ 0.7 and 1.3 $= z = 2$) the magnitude of the factor multiplying R is determined by linear interpolation between 0.4 and 4. We now take

$$x_{p+1} = x_p - (S_x^p \square \alpha I)^{-1}F_x^p \quad \text{or} \quad x_{p+1} = x_p - S^{-1}{}_x^p F_x^p$$

according to whether α is positive or not. If $\alpha = 0$ at this point, we generally directly computed the step size necessary to produce a positive α. This typically saved a number of iterations. Now x_{p+1} is the maximum of the quadratic approximation to the function on a region $B_{\pm}$ of radius

$$\|(S_x^p \square \alpha I)^{-1}F_x^p\|$$

with the center at x_p.

(b) F_x^p is so near 0 that the length of the step taken is within a preset tolerance of 0. Then, if S_x^p is negative definite, the process is terminated and x_p is accepted as the location of the maximum. If S_x^p is not negative definite, we are at

a saddle point or at the bottom of a cylindrical valley. A step is taken along the eigenvector corresponding λ_1 and the algorithm recycles in the usual manner. One final feature, incorporated for reason of computational efficiency rather than theoretical elegance, was the introduction of a scalar h_p into (2), writing it as

$$x_{p+1} = x_p - h_p(S_x^p \square \alpha I)^{-1} F_x^p.$$

At each step the computation is first performed with $h_p = 1$. If this gives an improvement in $H(x)$, h_p is multiplied by a constant, which magnitude is a decreasing function of the absolute value of the angle between the current step and the immediately preceding step. Then the function is examined at the new point so obtained. This process is repeated until the function declines in which event the last step is accepted. It should be noted that these attempts at stretching the step are relatively cheap since they require only an evaluation of the function. This is in contrast to changes in α within each iteration which require reinversion of $(S_x^p - \alpha I)$.

The following options of the algorithm must be set.

- *Calculation accuracy* is a positive double value to determine the difference between deviation of simulation result from experimental dataset under parameter values of current calculation step and values of previous calculation step. The default is '1.0E-5'.

Multi-objective cellular genetic algorithm (MOCell)

This algorithm [6] based on the cellular model of genetic algorithms. In such model the concept of (small) neighborhood is intensively used; this means that an individual may only interact with its nearby neighbors in the breeding loop. The overlapped small neighborhoods of the model help in exploring the search space because the induced slow diffusion of solutions through the population provides a kind of exploration (diversification), while exploitation (intensification) takes place inside each neighborhood by genetic operations. MOCell is an adaptation of a canonical cellular genetic algorithm to the multi-objective field. It uses an external archive to store the non-dominated solutions found during the execution of the algorithm. The main feature characterizing MOCell is that a number of solutions are moved back into the population from the archive after each iteration, replacing randomly selected existing individuals.

The following options of the algorithm must be set.

- *Iteration limit* is a maximum value of iterations. The default is '1500'.
- *Population size* is a number of the individuals in the population. The default is '20'.

Multi-objective particle swarm optimization (MOPSO)

The particle swarm optimization (PSO) algorithm, initially proposed by Kennedy and Eberhart [7], is a direct search algorithm based on the simulation of the social behavior of birds within a flock. The swarm is typically modeled by particles in the multidimensional space that have a position and a velocity. These particles fly through hyperspace and have two essential reasoning capabilities: their memory of their own best position and knowledge of the global or their neighborhood's best. Let $x_i(t)$ denote the position of particle p_i at time step t. The position of p_i is then changed by adding a velocity $v_i(t)$ to its current position, i.e.

$$x_i(t) = x_i(t - 1) + v_i(t).$$

Let *gbest* is the position of the best particle from the entire swarm and *pbest* is the position of the neighborhood best that the particle obtained by communicating with a subset of the swarm. In this case, velocity equation is given by

$$v_i(t) = W \cdot v_i(t - 1) + C_1 r_1 (x_{pbest} - x_i(t)) + C_2 r_2 (x_{gbest} - x_i(t)),$$

where W is the inertia weight, C_1 and C_2 are the learning factors (usually defined as constants), and r_1, r_2 are random values from the interval [0,1]. Here we used a multiple-objective particle swarm optimizers based on the paper of Sierra and Coello [8]. This optimizer allows to take into account parametric constraints and uses a crowding factor for the leaders selection as well as the following mutation operators: an uniform mutation operator, where the variability range allowed for each decision variable is constant over generations, and a non-uniform mutation operator, where such range decreases over time. These operators modify the values of the particle decision variables with a certain probability.

The following options of the algorithm must be set.

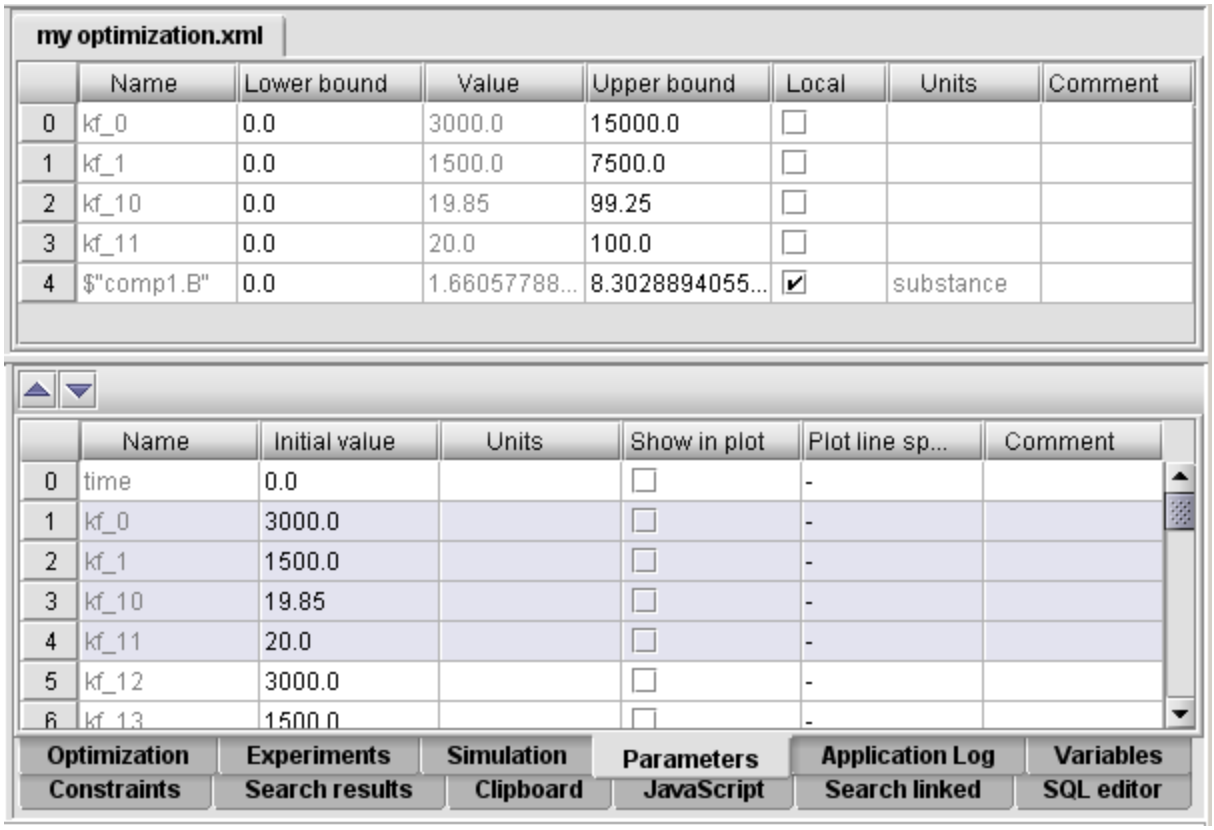
- *Number of iterations* is a maximum value of iterations. The default is '500'.
- *Number of particles* is a number of the particles in a swarm. The default is '20'.

References

1. L Ingber (1996), Adaptive simulated annealing (ASA): Lessons learned, *Control and Cybernetics*, **25(1)**:33-54.
2. TP Runarsson, X Yao (2000), Stochastic Ranking for Constrained Evolutionary Optimization, *IEEE Transactions on Evolutionary Computation*, **4(3)**:284-294.
3. M Björkman, K Holmström (1999), Global Optimization Using the DIRECT Algorithm in Matlab, *AMO*, **1(2)** :17-37.
4. DR Jones (2001), DIRECT global optimization algorithm. in: *Encyclopedia of optimization*, 431-440.
5. SM Goldfeld, RE Quandt, HF Trotter (1965), Maximization by Quadratic Hill-climbing, *Econometric Research Program, Research Memorandum* **72**.
6. AJ Nebro, JJ Durillo, F Luna, B Dorronsoro and E Alba (2009), A Cellular Genetic Algorithm for Multiobjective Optimization, *International Journal of Intelligent Systems*, **24(7)**: 726-746.
7. J Kennedy, RC Eberhart (1995). Particle Swarm Optimization, *Proceedings of the 1995 IEEE International Conference on Neural Networks*, Piscataway, New Jersey, IEEE Service Center, 1942-1948.
8. MR Sierra, CA Coello Coello (2005), Improving PSO-Based Multi-objective Optimization Using Crowding, Mutation and e-Dominance, *Evolutionary Multi-Criterion Optimization*, **3410**:505-519, Springer Berlin/ Heidelberg.

9.1.3 Fitting parameters

Before the optimization will be performed you need to select the parameters to fit. To do this open the tab with the diagram parameters (variables), select the parameters (variables) which you want to fit, and click the up arrow. The list of the selected parameters appear in the table located above.



In order to remove unnecessary parameters from the fitting parameters set, select them in the table and click the down arrow. If you want to change the initial values of some parameters (variables), enter them in the column "Initial value" of the table with the diagram parameters (variables). The values of the relevant parameters (variables) are automatically changed in the fitting parameters table.

Fitting parameters can be specified as local or global to work with experimental data obtained for the different cell lines. For more details see Experimental data chapter.

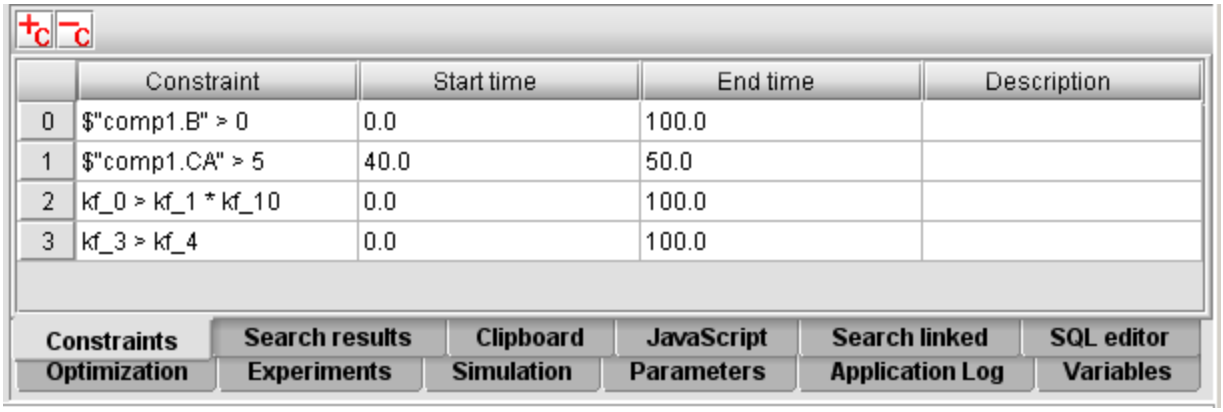
9.1.4 Parameter constraints

Constrained optimization is the minimization of an objective function subject to constraints on the possible values of the independent variable. Constraints can be either equality or inequality constraints and are specified in the special tab of the optimization document with time intervals in which they must be fulfilled. During the optimization of diagram parameters under the given constraints a penalty function is calculated to establish a measure of their violation and this function value is strived to the minimum. In the region where constraints are not violated this value is equal to zero. In our implementation the penalty function is determined by the formula

$$P = ?_k \max(0, g_k(x))^2,$$

where x is a vector of diagram parameters and variables, and $g_k(x)$ d 0 are the parameter constraints.

Constraints can be indicated in the tab "Constraints" of the optimization document. Remember when you write a constraint you can use only the identifiers of parameters and variables declared in the diagram.



To add a new parameters constraint, press the button with the symbol plus on it. To remove some constraints, select them and press the button with the symbol minus.

9.1.5 Experimental data

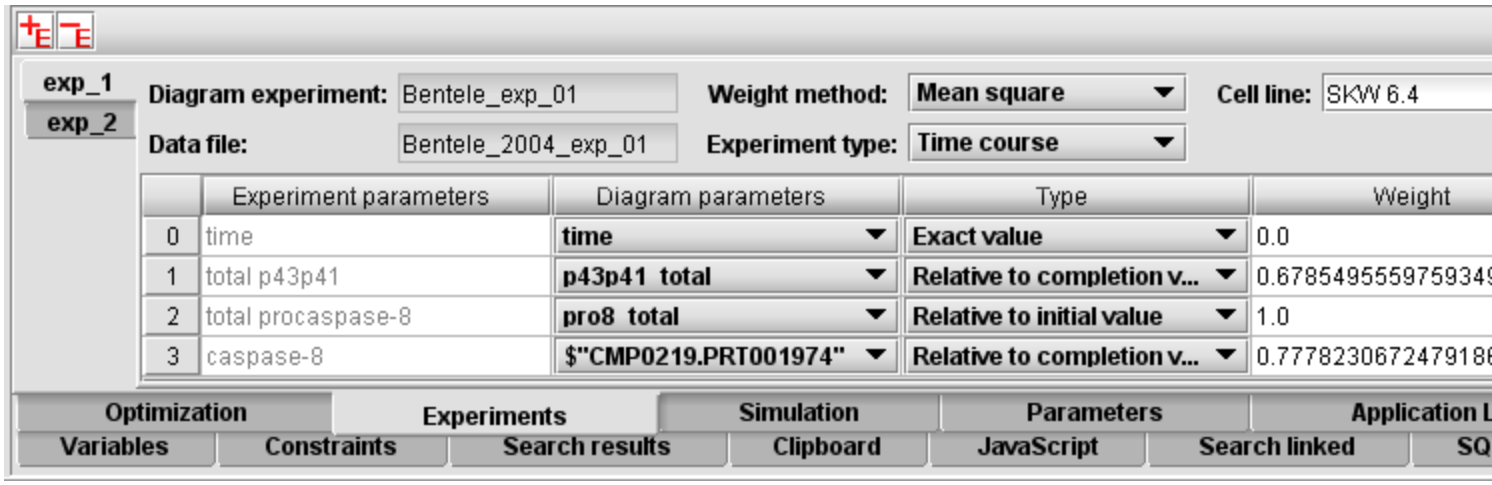
Experimental datasets must be the results of time courses expressed as exact or relative values of substance concentrations and may be comprised of several files each including possibly multiple experiments.

The diagram parameters estimation can not be perform until the experimental data are not set. Optimization experiments can be process in the "Experimental data" tab. To add a new experiment, press the button with the symbol plus on it. To define the experiment you can fill the following fields.

- *Name*. The optimization experiment name (must be unique).
- *Diagram experiment*. The diagram experiment name, which is specified when you need to fit parameters of the diagram modification identified by the diagram experiment. If the value is equal to the empty string the fitting is performed for the diagram without changes.
- *Data file*. The name of the file with experimental data. The data must be stored in folder BioUML\data_resources\Optimization\Experiments.
- *Weight method*. The weight method to make all trajectories of each variable have similar importance in the fit. Valid methods for the weight w_j calculation are mean: $w_j = 1/|\langle x_j \rangle|$, mean square: $w_j = (\langle x_j^2 \rangle)^{-1/2}$, and standard deviation: $w_j = 1/|\langle x_j^2 \rangle - \langle x_j \rangle \langle x_j \rangle|$.
- *Experiment type*. The type of experimental data is time course or steady state.
- *Cell line*. The parameter for separation of the experiments on the groups. All the fitting parameters declared as local remain global within the same cell line. For the experiments where cell line value is the empty string all local parameters are independently fitted.

You also need to specified the array of connections between diagram variables and columns of the experimental data table. For this purpose fill the table automatically generated in the experiment tab. The following fields must be identified.

- *Experiment parameters*. The experimental table column names.
- *Diagram parameters*. The diagram parameters or variables.
- *Type*. The type of the experimental data column: exact value, relative to initial value or relative to completion value.
- *Weight*. The weight automatically calculated for each data column. It can be changed by the user.



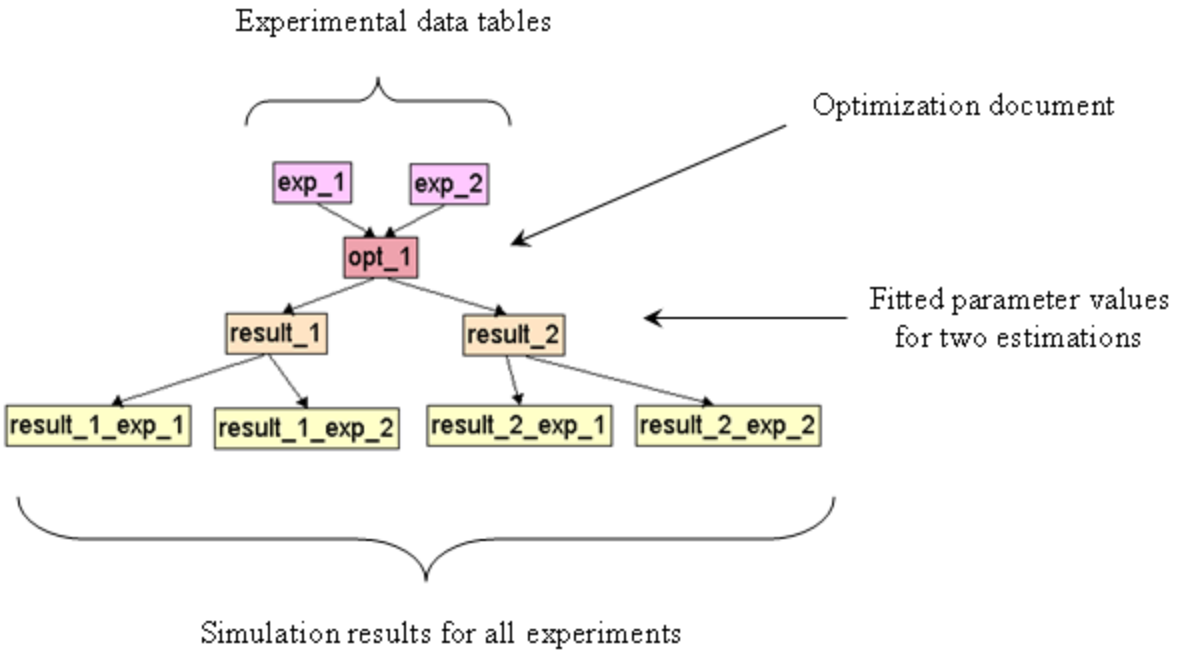
To remove some experiments, select them and press the button with the symbol minus.

Example of the experimental data file contents

```
ID              Diagram ID
OS
PLATFORM
TITLE           Experimental data title
DESCRIPTION     Experimental data description..
SAMPLES
#1      time; Float; NONE; v1
#2      Substance ID; Float; NONE; v2
PROPERTIES
DATA
0       0.0     0.01
1       50.0    0.36
//
```

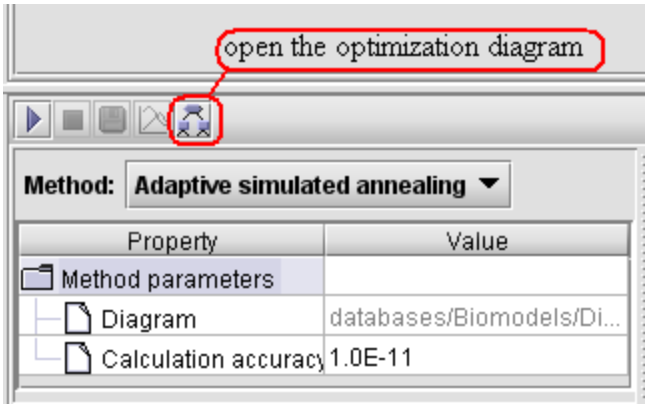
9.1.6 Optimization diagram

In order to facilitate the work with all the data used and obtained during the optimization we have developed a special type of the optimization diagram, where the set of vertices corresponds to the operational data, and the arcs show the relations between these data elements.



After the double click on such diagram elements corresponding documents with experimental data tables, fitted parameter values or simulation results plots will be opened.

The optimization diagram is automatically generated during the optimization document creation and changes. You can find it in the tree under "Data" -> "Optimization" -> "Diagrams". To open the optimization diagram from the optimization document click the open button in the document "Method" tab.



10 Genome browser

Genome browser is used to display sequences with several tracks like one document. BioUML allows to combine sequences and tracks from different databases in one document. For example, it is possible to get sequence from UCSC database and add Ensembl tracks to it.

To create new document select sequence in repository tree and select **Open as project** in context menu. Select necessary tracks and press **Ok**.

NOTE: tracks from other database is available with Tracks view part later.

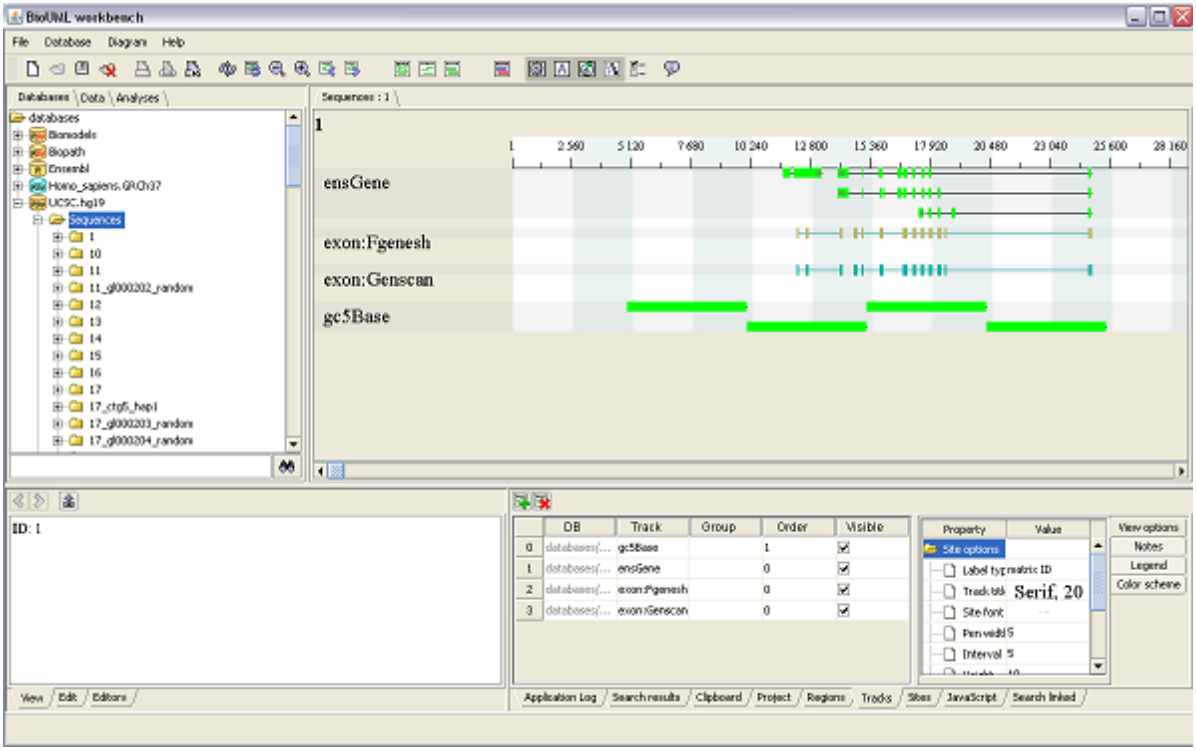


Figure 10.1. Genome browser overview.

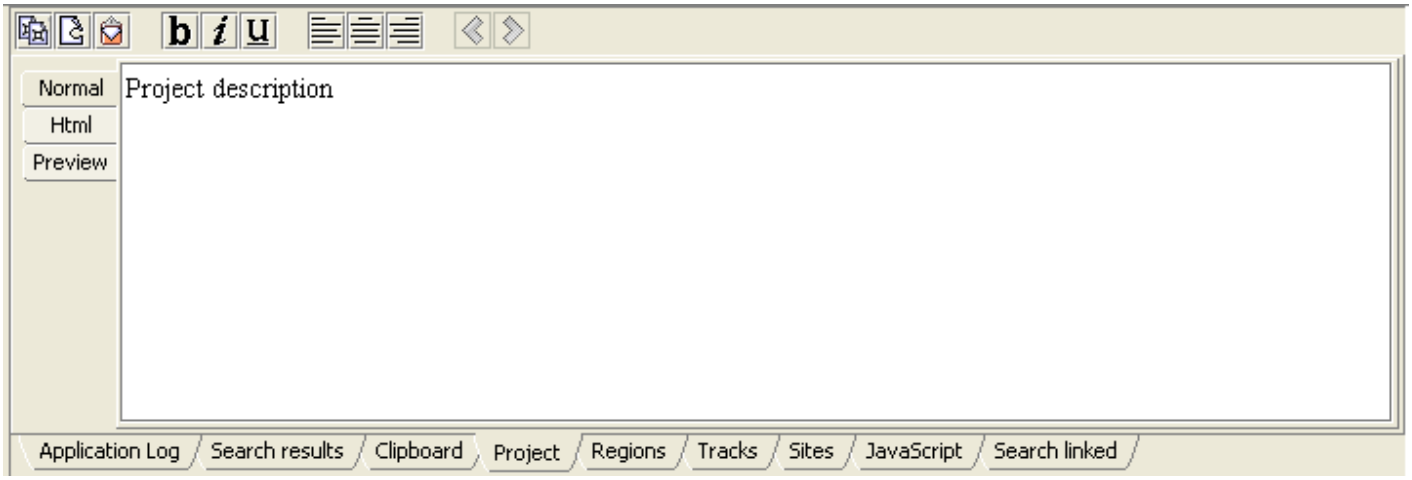
View parts

View parts are used to configure document model and view. There are several special view parts for genome browser:

- Project
- Regions
- Tracks
- Sites

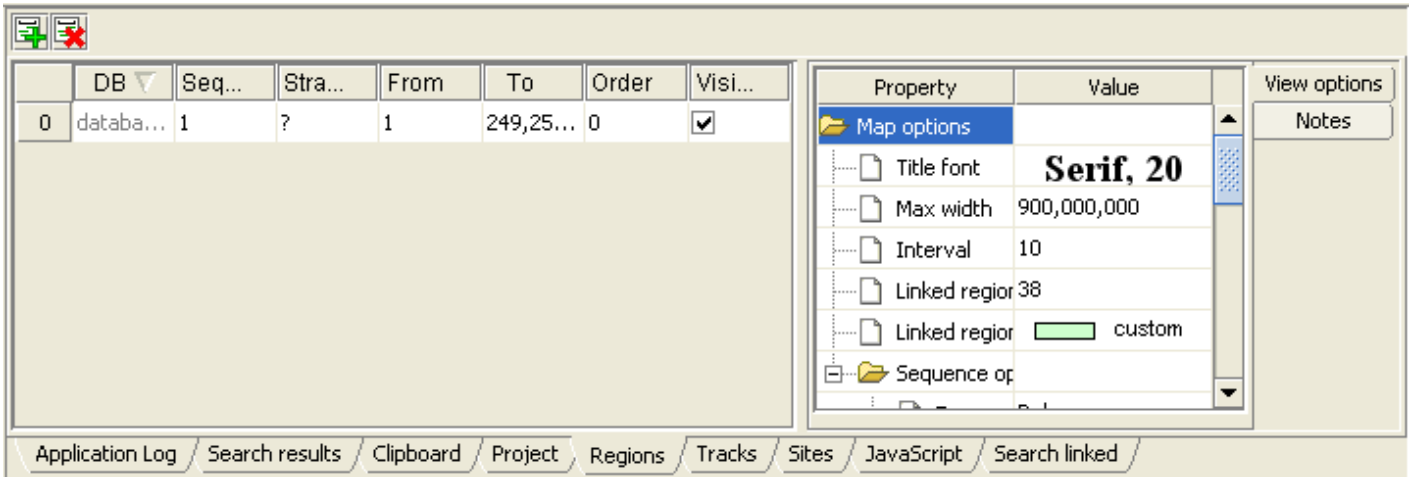
10.1 Project

Project view part allows to view and edit project description.



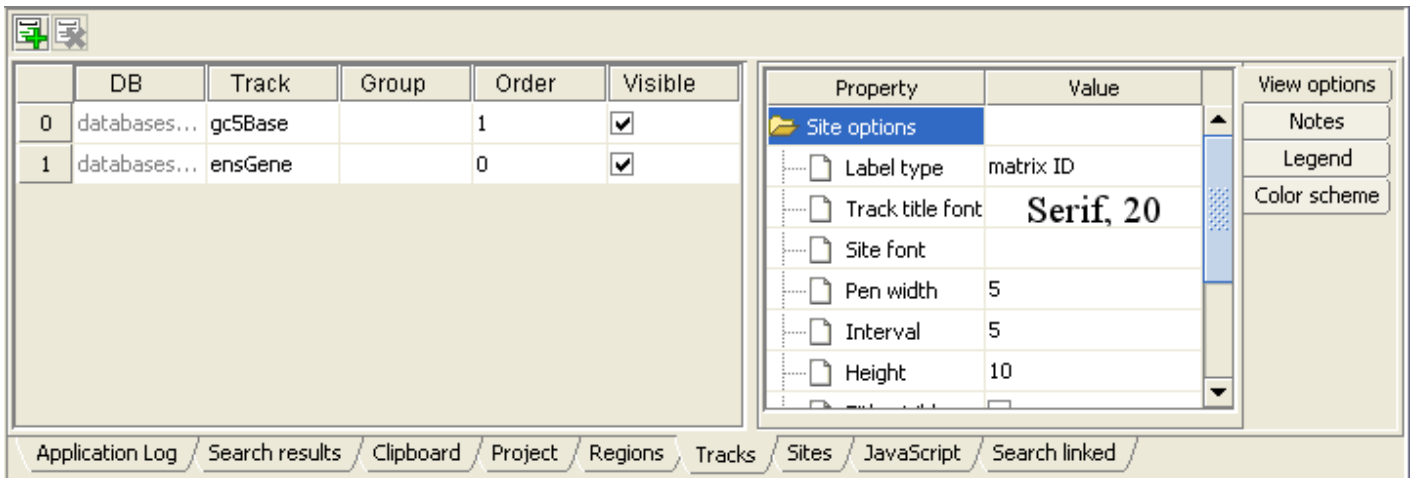
10.2 Regions

Region is the part of sequence. It is possible to use several sequences parts in one document. **Regions** view part allows to manage project sequences and it's view options.



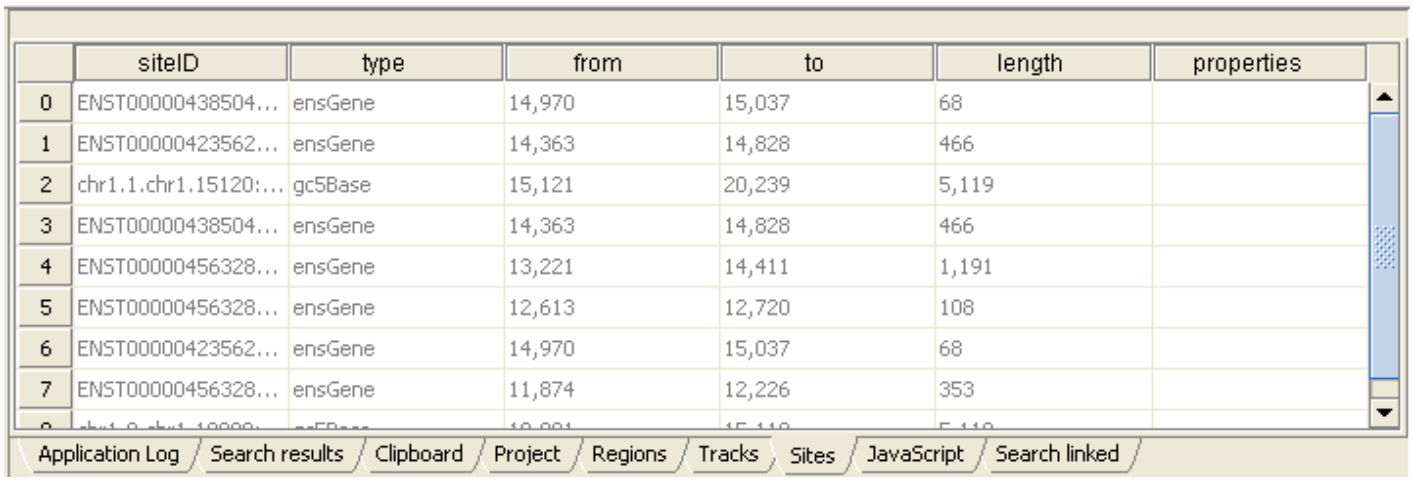
10.3 Tracks

Tracks view part allows to manage tracks for document(add/remove track, set visibility, and display order) and it's view options and color schemes.



10.4 Sites

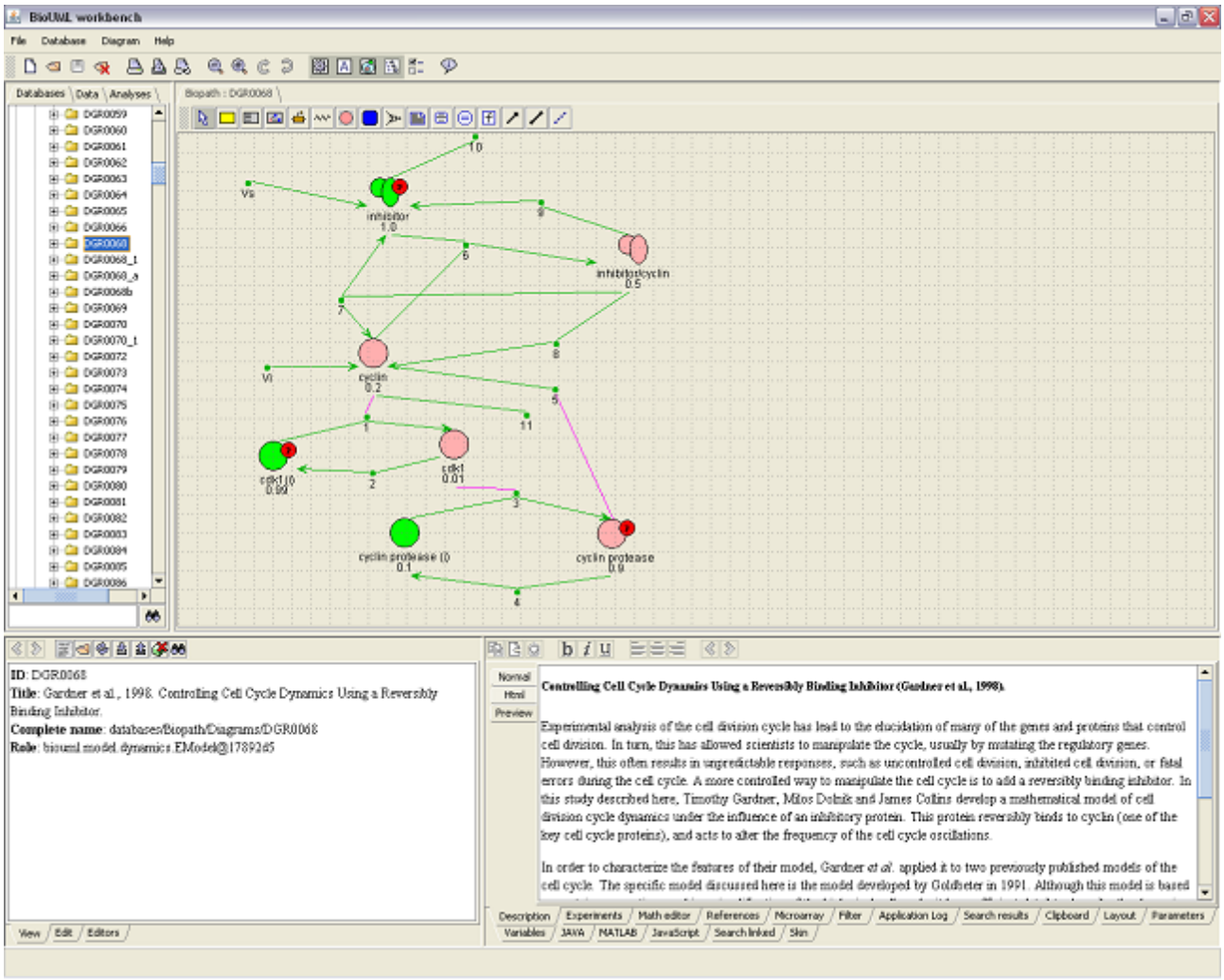
Sites view part is used to display detailed information about visible sites.



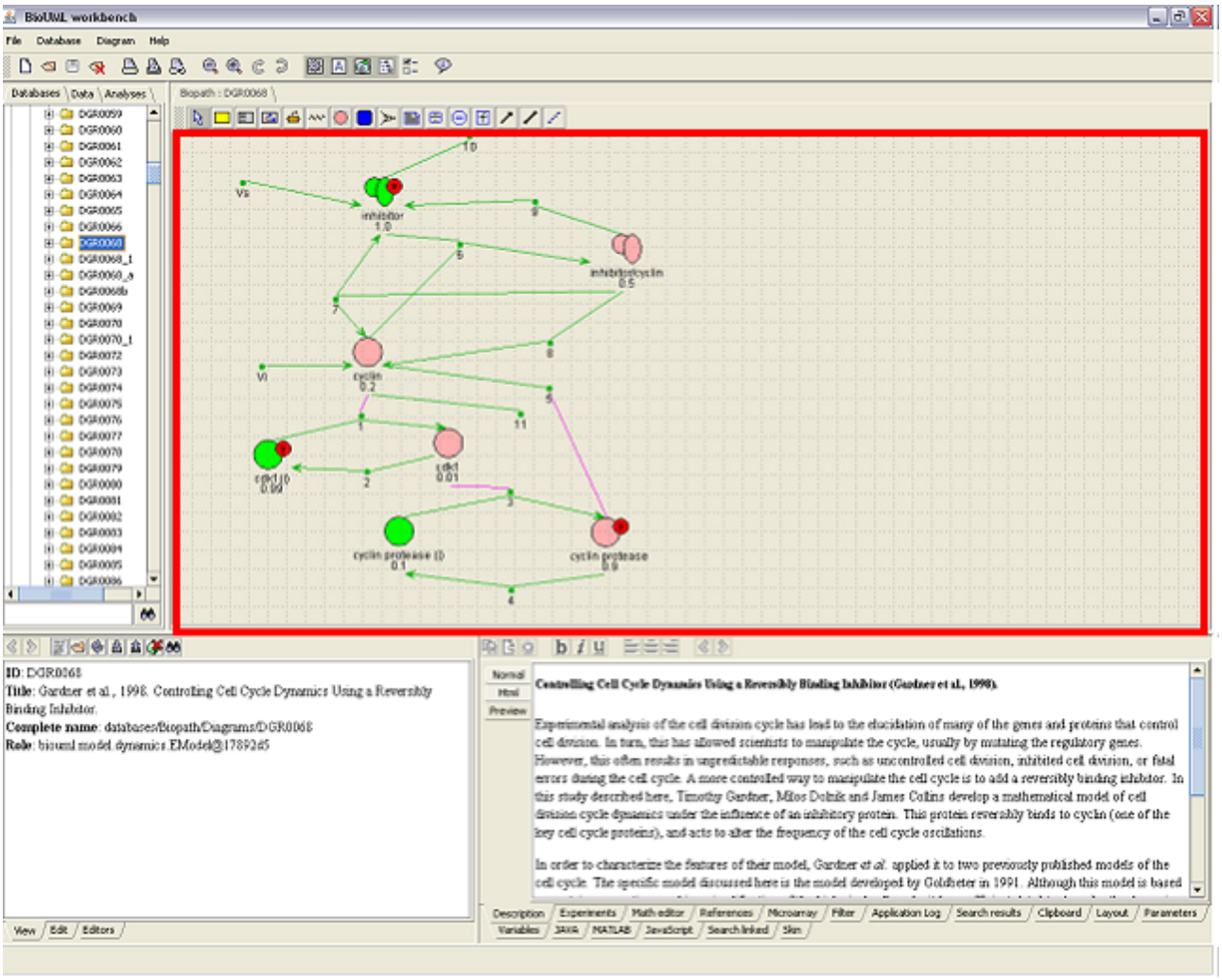
11 Simulation

The process of model simulation and its adjustment can be demonstrated on the example of the cell cycle model (DGR0068 in BMOND/Biopath; Gardner et al., 1998).

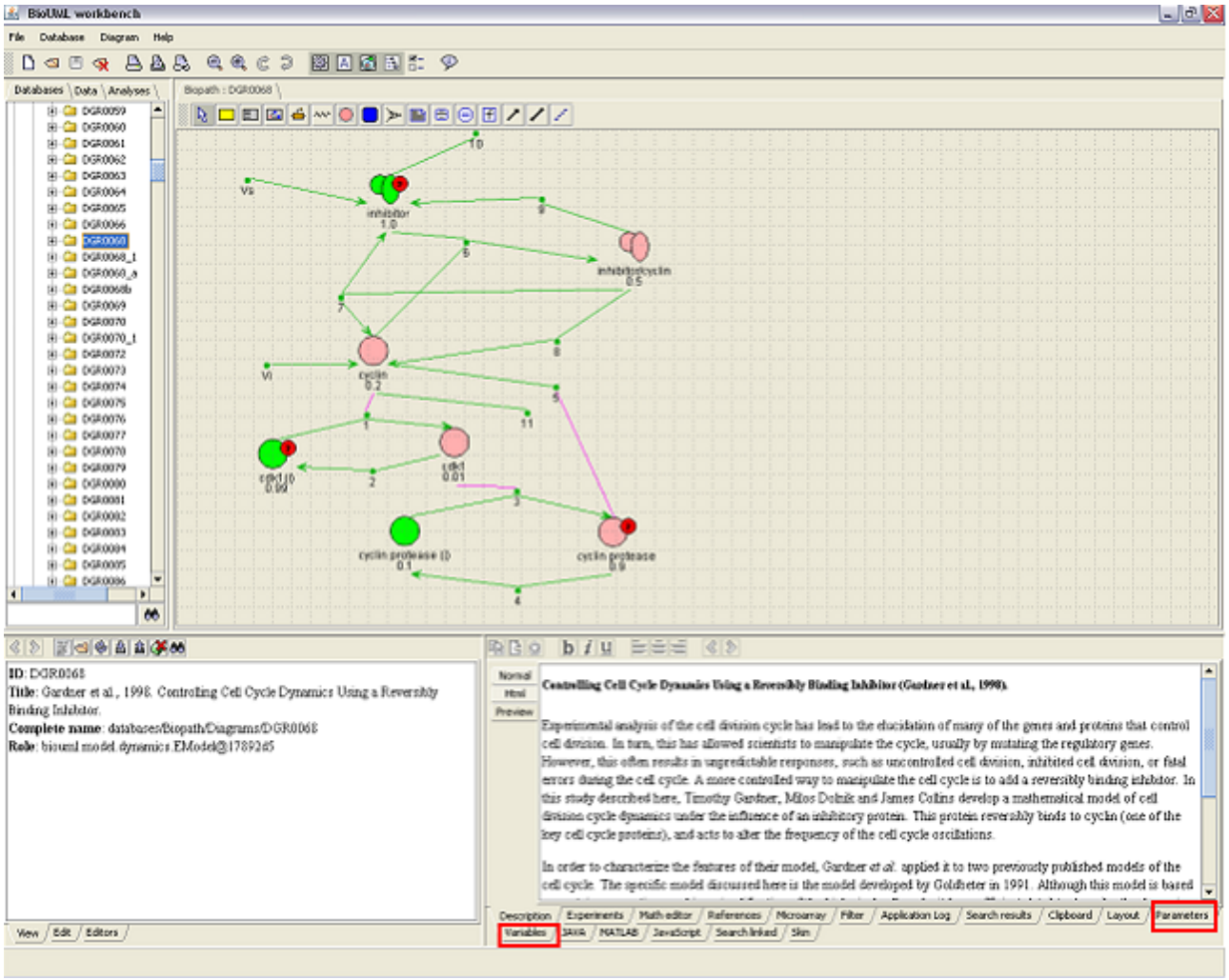
After you opened a diagram-model you will see the view like this:



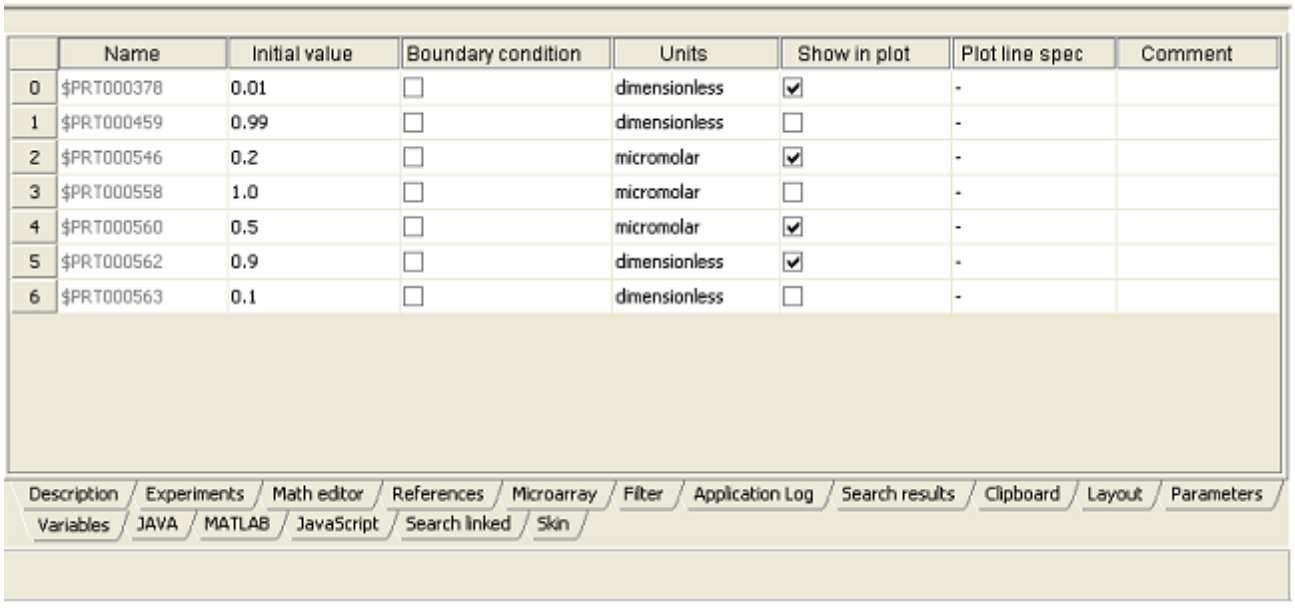
The upper right part of the window contains the scheme of the model with proteins, biochemical interactions (arrows) and set initial values of protein (or something else) concentrations.



The model consists of variables (protein concentrations), constants (the constants of included biochemical reactions), and various additional conditions (events, mathematical functions, etc not shown in this model). Values of variables and constants can be observed in the menu Variables and Parameters (a range of panels at the bottom of the lower right part of the window), respectively.



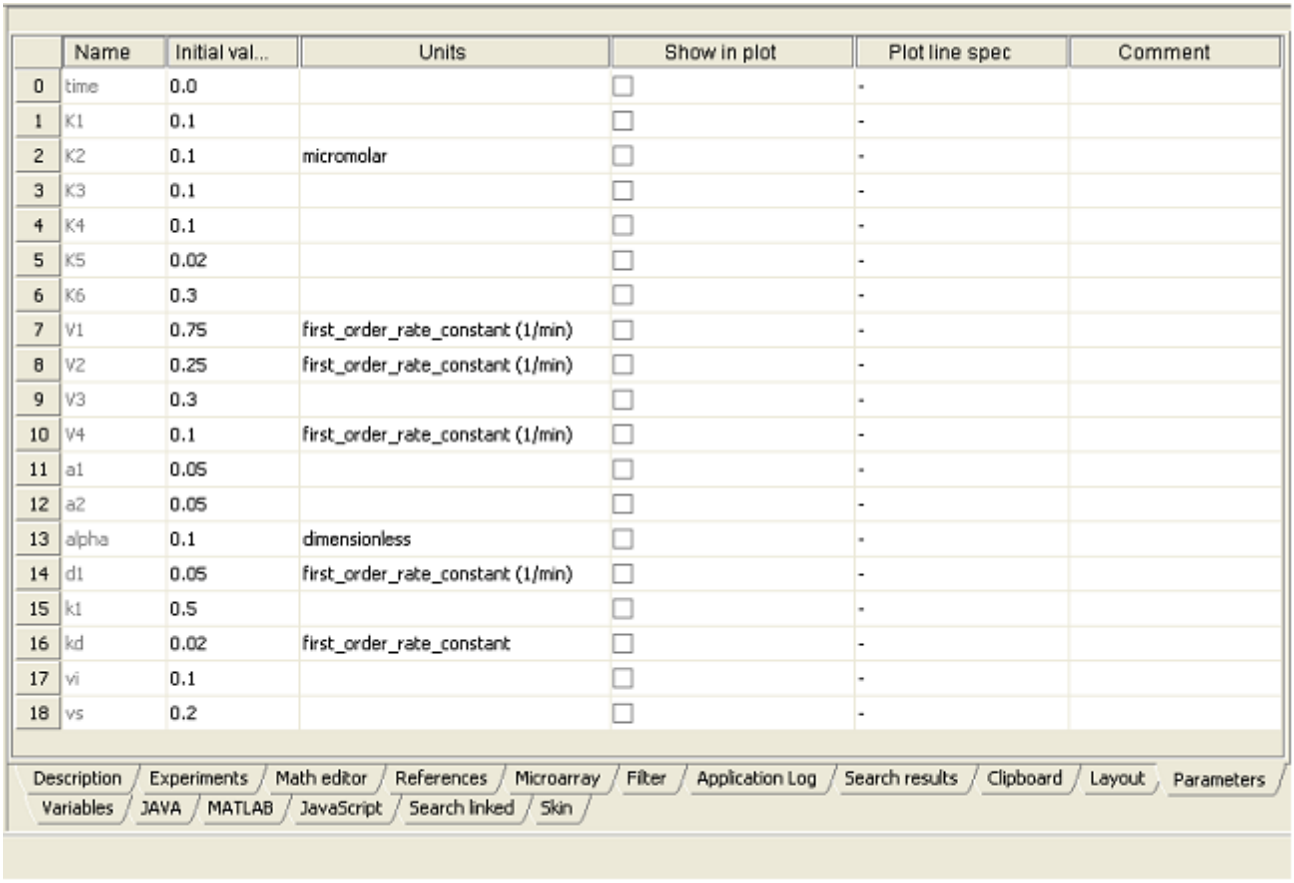
By clicking those menus you will get the list of desirable entities with respective values (as it is shown below for variables of this model).



	Name	Initial value	Boundary condition	Units	Show in plot	Plot line spec	Comment
0	$PRT000378	0.01	☐	dimensionless	☑	-	
1	$PRT000459	0.99	☐	dimensionless	☐	-	
2	$PRT000546	0.2	☐	micromolar	☑	-	
3	$PRT000558	1.0	☐	micromolar	☐	-	
4	$PRT000560	0.5	☐	micromolar	☑	-	
5	$PRT000562	0.9	☐	dimensionless	☑	-	
6	$PRT000563	0.1	☐	dimensionless	☐	-	

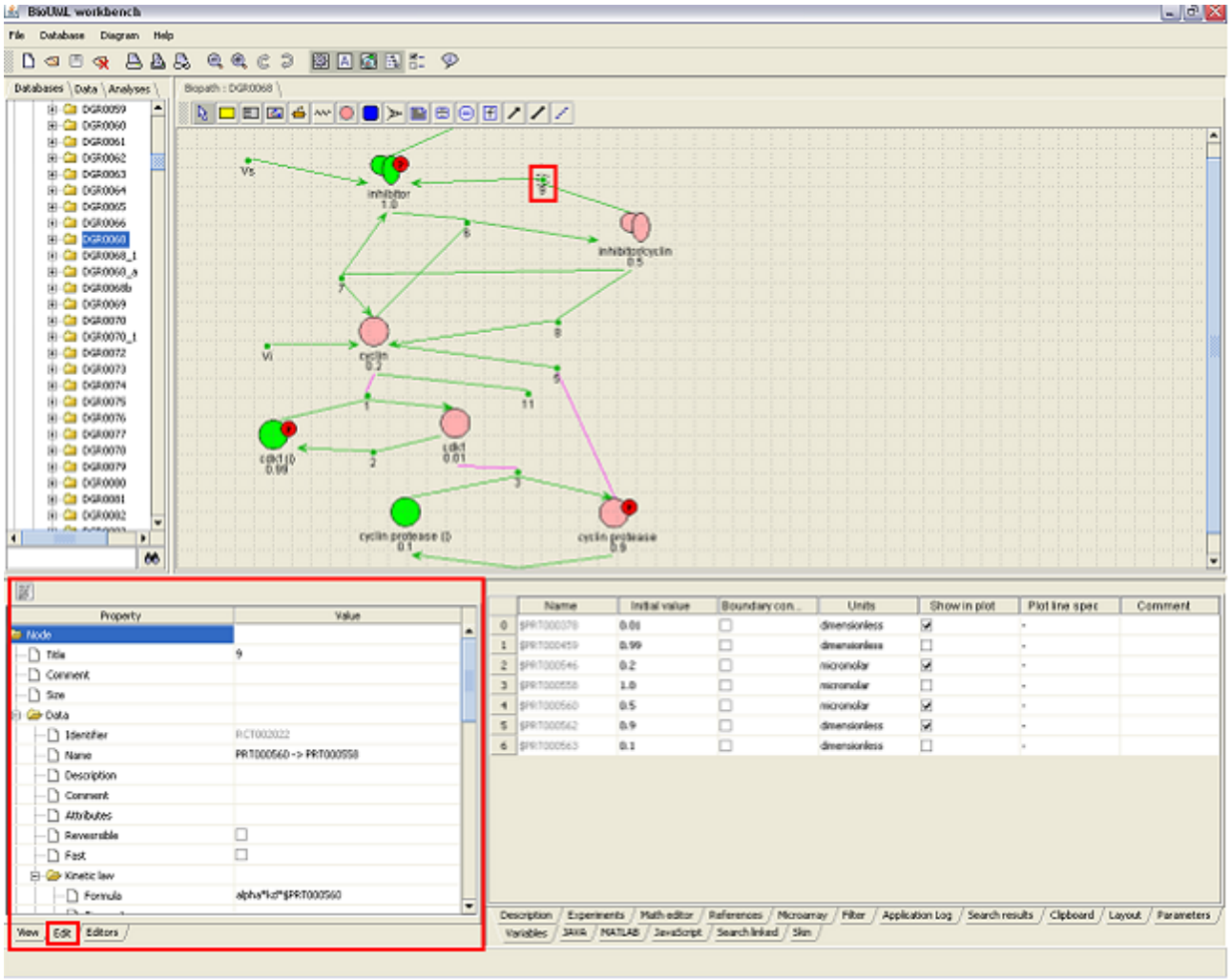
Description Experiments Math editor References Microarray Filter Application Log Search results Clipboard Layout Parameters
Variables JAVA MATLAB JavaScript Search linked Skin

The column "Name" contains database IDs of the components of the model that have to be used in all formulas and equations describing the dynamics of the concentration changes. Initial values of variables can be changed by clicking of respective table cell and input of desirable value using keyboard. "Boundary condition" can be used by ticking respective table cell to fix the concentration of a variable in the process of model simulation. "Units" contains dimension data. "Show in plot" allows to regulate how many graphs (and variables, respectively) should be demonstrated during simulation on the plot. The same logic is used for "Parameters" menu (see below).

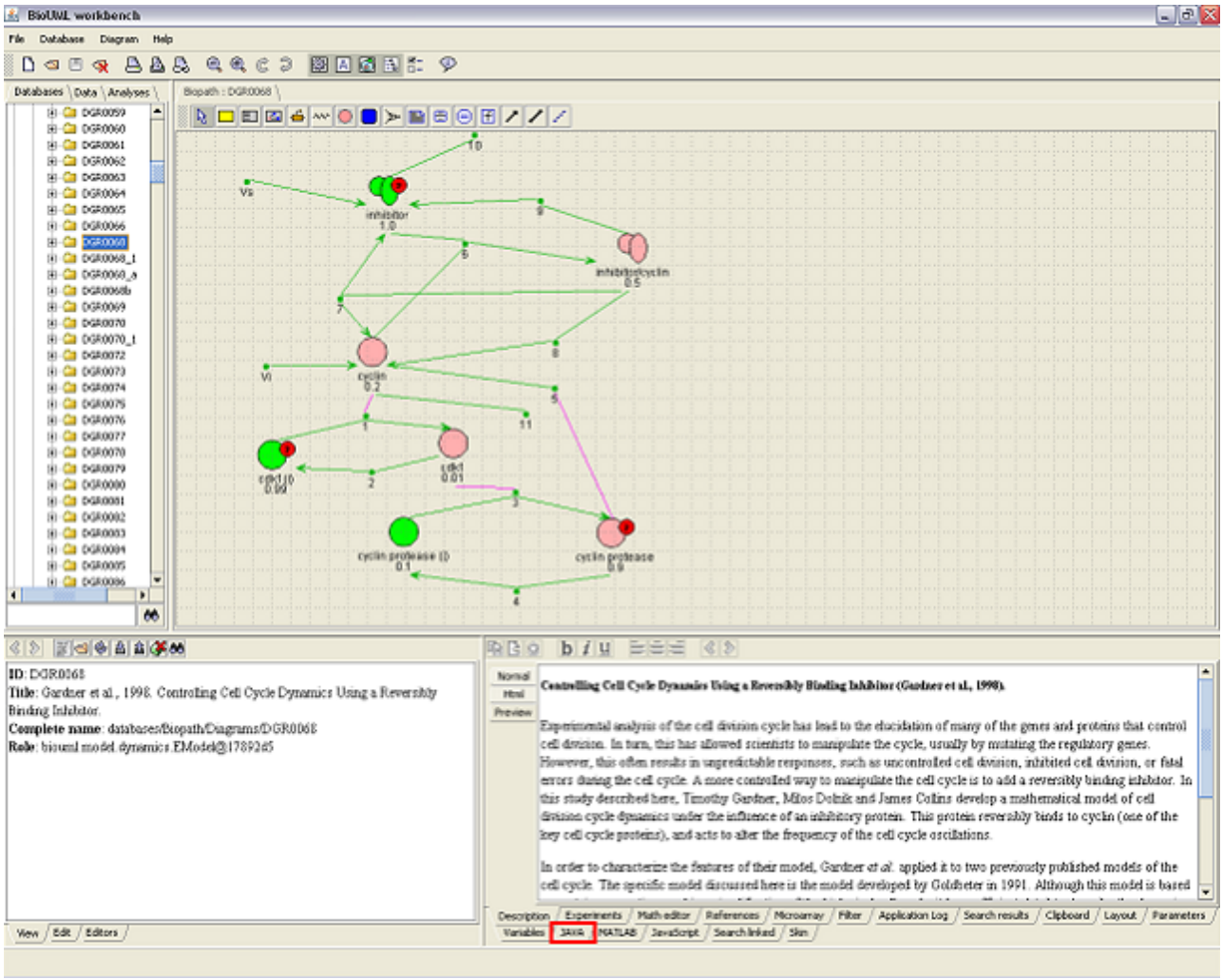


	Name	Initial val...	Units	Show in plot	Plot line spec	Comment
0	time	0.0		☐	-	
1	K1	0.1		☐	-	
2	K2	0.1	micromolar	☐	-	
3	K3	0.1		☐	-	
4	K4	0.1		☐	-	
5	K5	0.02		☐	-	
6	K6	0.3		☐	-	
7	V1	0.75	first_order_rate_constant (1/min)	☐	-	
8	V2	0.25	first_order_rate_constant (1/min)	☐	-	
9	V3	0.3		☐	-	
10	V4	0.1	first_order_rate_constant (1/min)	☐	-	
11	a1	0.05		☐	-	
12	a2	0.05		☐	-	
13	alpha	0.1	dimensionless	☐	-	
14	d1	0.05	first_order_rate_constant (1/min)	☐	-	
15	k1	0.5		☐	-	
16	kd	0.02	first_order_rate_constant	☐	-	
17	vi	0.1		☐	-	
18	vs	0.2		☐	-	

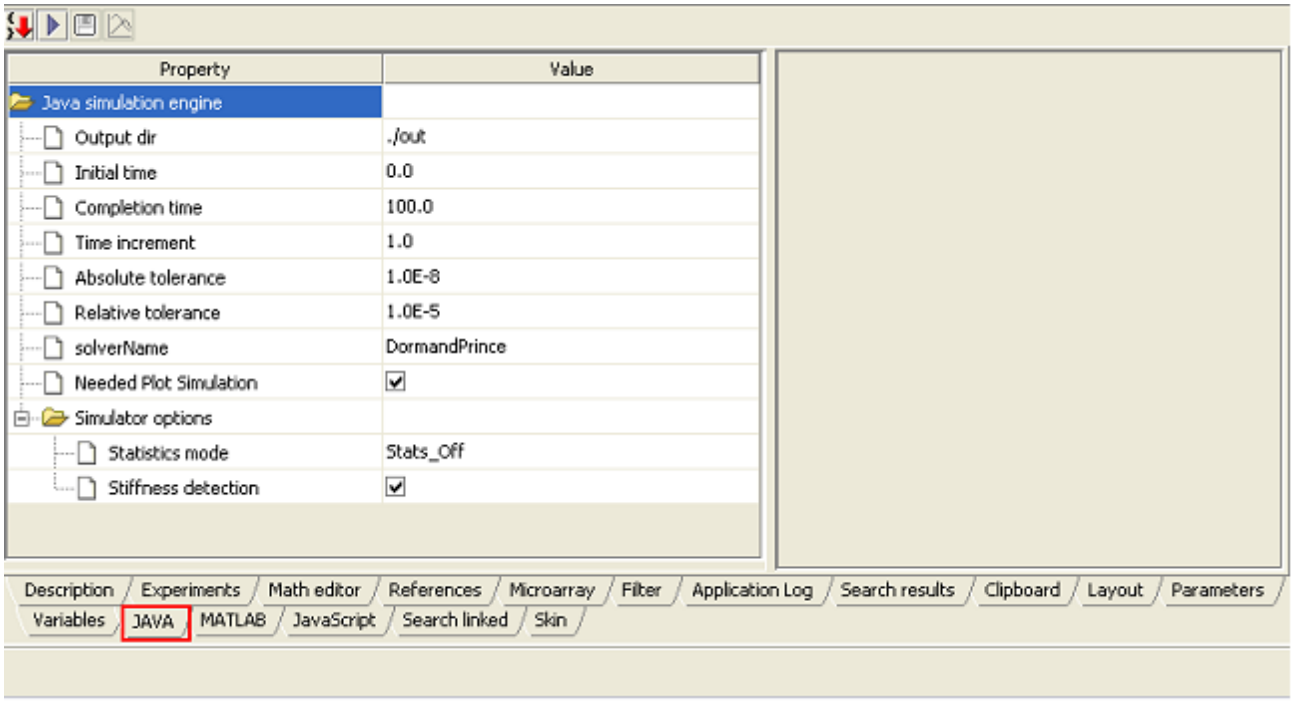
Biochemical formulas are one of the important components of the model. You can add them during creation of biochemical reactions using BioUML tools or add/edit by clicking a dot on an existing arrow (as it is shown on the example of the reaction number 9) and using "Edit" menu of the lower left part of the window like it is shown below. In the chapter "Kinetic law" you will see the formula or can add new one if there is no any formula. Newly entered variables and constants will be added to the respective lists "Variables" and "Parameters" automatically.



After all values were inputted and all components of the model were confirmed you can go to the menu "Java" containing tools for adjustment of simulation procedure.



You will see the window like demonstrated below containing info about predefined parameters of simulation process:



"Output dir" – the destination folder for saving simulation results.

"Initial time" (set 0.0) – here you can choose the time point starting from you will see the results of simulation as graphs.

"Completion time" – the time period for which you will simulate the changes of protein concentrations, etc. (may be given in seconds, minutes, etc according to the dimensionality of constants).

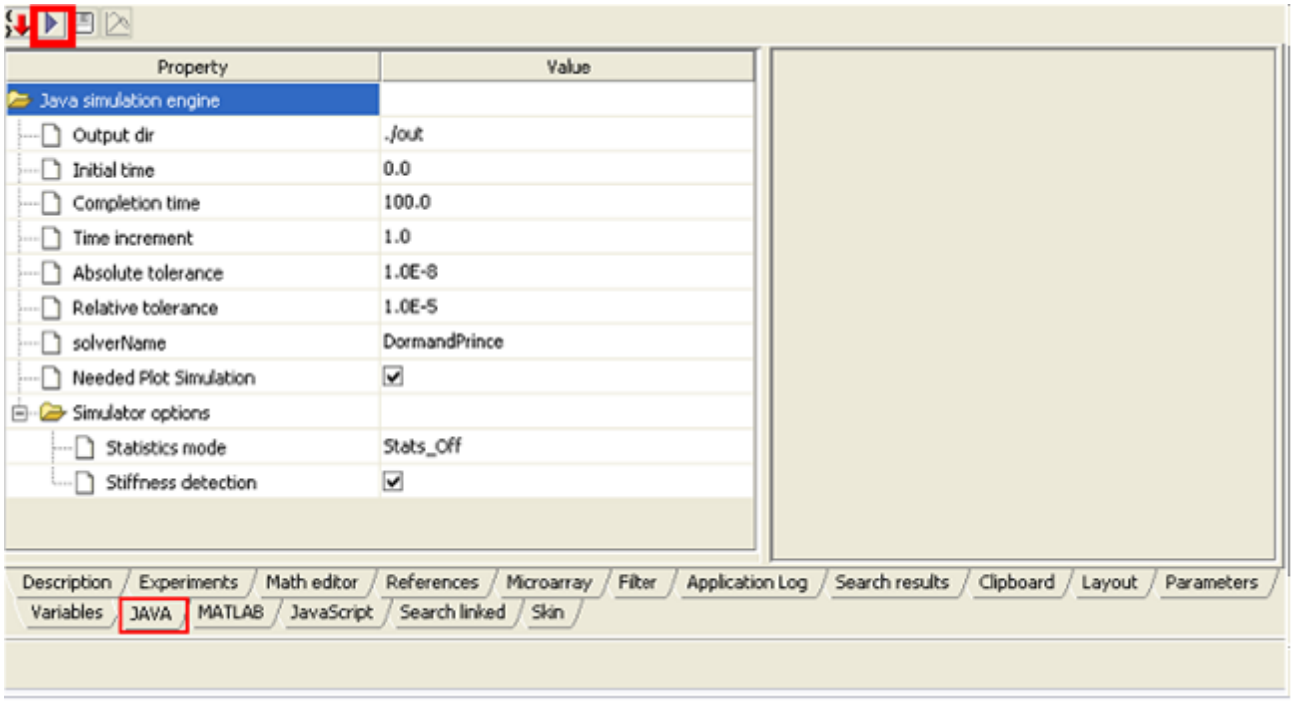
"Time increment" – the delta for time; the less increment the more calculations will be made by a solver to get values of variables (and the graph will be smoother), but it is time consuming. The bigger increment, the faster calculations and the graph will be rough. Default value is 1.0, you can use 0.1, for example.

"Absolute tolerance" and "Relative tolerance" – these parameters are used to select the time interval for calculations.

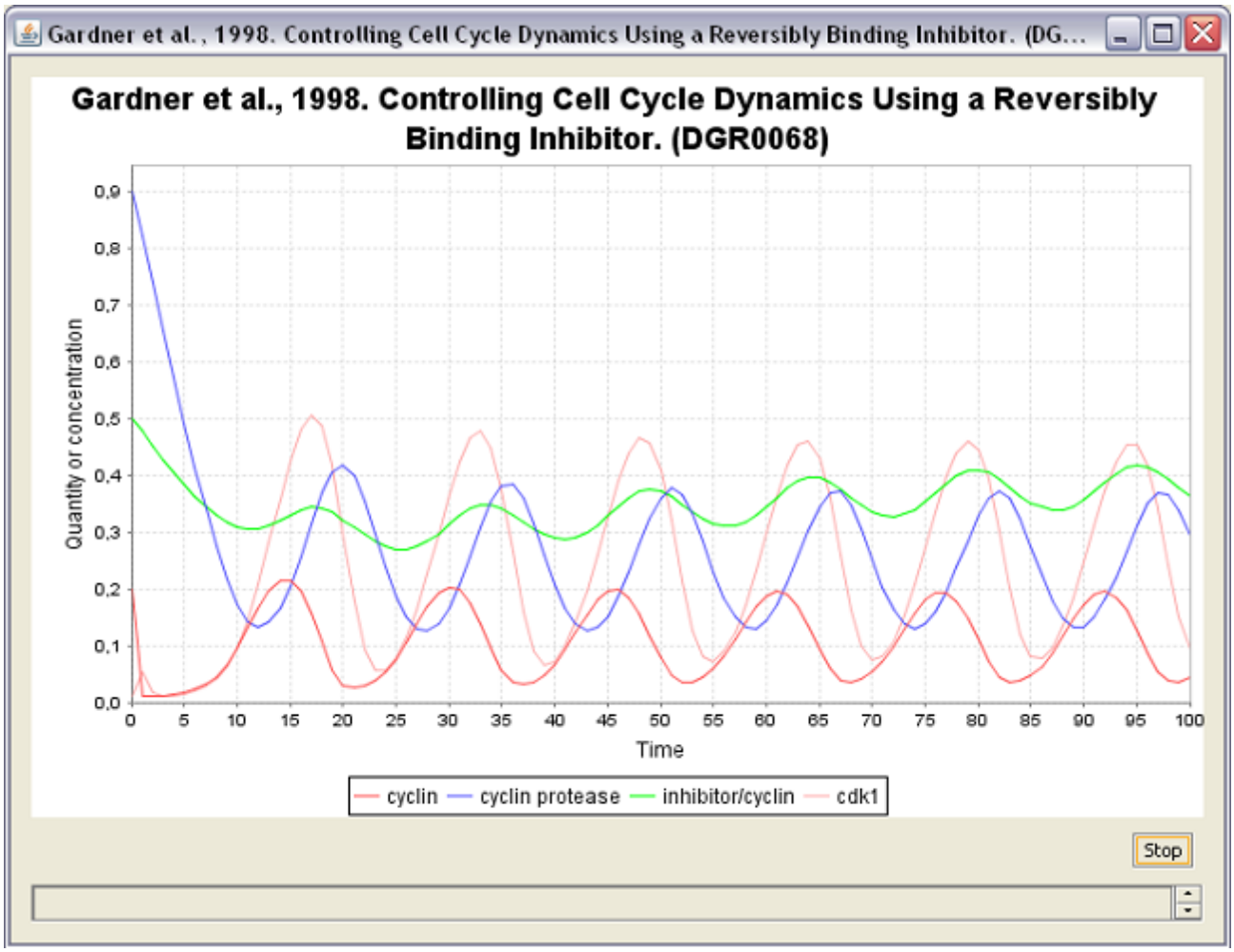
"Solver name" – here you can select a solver for calculations and simulation of results.

"Needed plot simulation" – if selected then you will see a plot with simulation results on it in the regime of real time.

After you confirmed all parameters, you can start simulation process by clicking button "Play" (as it is shown below).



In new window you will see a plot with graphs reflecting changes of variable values in time.



Clicking buttons depicted below (red) will allow to save results (calculated values for each time point regarding to the time increment value) and show the window with plot to modify output graphs (remove undesirable and so on).

For solving ordinary differential equation initial value problem next solvers were applied in BioUML:

- Euler
- Dormand-Prince
- JVODE
- RADAU5

11.1 Euler

Explicit Euler method for solving ODE initial value problem. Does not support event location or stiffness detection. In general, usage of this method can not be recommended.

User-provided solver parameters:

- *Initial step.* Time step for method, it does not change while integration.

11.2 Dormand-Prince

Explicit Runge-Kutta scheme known as the Dormand-Prince routine, which is 5th order/4th order, a 7-stage method that can solve for an ODE. It also features stiffness detection and event location. Could only be used for solving ODE with non-stiff systems.

Currently Dormand-Prince method is the default method for solving ODE in MATLAB.

REFERENCES:

Dormand, J. R.; Prince, P. J. (1980), A family of embedded Runge-Kutta formulae, *Journal of Computational and Applied Mathematics* **6 (1)**: 19–26

11.3 JVODE

Java Variable-Coefficient ODE (JVODE) solver is a ported to java version of CVODE - part of a software family SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic equation Solvers developed by Lawrence Livermore National Library. JVODE solves ordinary differential equations initial value problems with either stiff and non-stiff systems. It was refactored from CVODE into one ODE solver and user can specify which integration method will be used.

JVODE supports events which are given by function g(y,t) satisfying the following conditions:

$g(y,t) < 0$ if event has not happened yet

$g(y,t) = 0$ in exact point of event

$g(y,t) > 0$ after event happened.

User-provided solver parameters:

- *Integration method.* Two linear multistep implicit methods are available:
    1. **Adams - Moulton** (recommended for non-stiff systems),
    2. **Backward Differential Formula** (recommended for stiff systems).
- *Inner linear solver type.* Method for solving non-linear equation on each time step. Available methods are:
    1. **Functional iterations.**
    2. **Newton iterations** (Using linear equation system solving and Jacobian approximation).
- *Jacobian approximation type.* In Newton iterations case user should also define type for Jacobian approximation:
    1. **Dense,** (recommended)
    2. **Banded,**
    3. **Diagonal.**

    User-provided Jacobian is not supported yet.
- *Mu, Ml.* Additional parameters for banded Jacobian approximation only. Note Mu and Ml must be such that: $0 < Mu,Ml < N$, where N is system dimension.

References

1. P.N.Brown, G.D. Byrne, and A.C. Hundmarsh, VODE, a Variable-Coefficient ODE Solver, *SIAM J. Sci. Stat. Comput.*, **10** (1989), pp. 1038-1051

2. S. D. Cohen, A.C. Hindmarsh, CVODE, A Stiff/Nonstiff ODE Solver in C.

11.4 RADAU5

This algorithm is ported for Java from C++ code (last updated: Nov 15, 2002), written by Blake Ashby (E-mail: bmashby@stanford.edu) on the base of code RADAU5 originally written in FORTRAN (version of July 9, 1996, latest small correction: January 18, 2002) by: E. Hairer and G. Wanner (Universite de Geneve, Dept. de Mathematiques Ch-1211 Geneve 24, Switzerland, E-mail: ernst.hairer@math.unige.ch, gerhard.wanner@math.unige.ch)

RADAU5 is in detail described in the book:

E. Hairer and G. Wanner, Solving Ordinary Differential
Equations II. Stiff and Differential-Algebraic Problems.
Springer Series in Computational Mathematics 14,
Springer-Verlag 1991, Second Edition 1996.

This code computes the numerical solution of a differential system of first order ordinary differential equations

$$y'=f(x,y).$$

The method used is an implicit Runge-Kutta method (RADAU IIA) of order 5 with step size control and continuous output. (See Section IV.8 of Hairer and Wanner). The method is designed to solve stiff systems, for non-stiff systems it is rationally to use simpler methods (requiring less computational load), e.g. Dormand-Prince.

User-provided solver parameters:

- **itoler**

itoler = 0: Both rtoler and atoler are scalars. The code keeps, roughly, the local error of y[i] below rtoler*fabs(y[i])+atoler
itoler = 1: Both rtoler and atoler are vectors. The code keeps the local error of y[i] below rtoler[i]*fabs(y[i])+atoler[i].

- **hinit**

Initial step size guess;
For stiff equations with initial transient, h = 1.0/(norm of f'), usually 1.0e-3 or 1.0e-5, is good. This choice is not very important, the step size is quickly adapted. (if h = 0.0 on input, the code sets h = 1.0e-6).

- **hmax**

Maximal step size, default (when hmax is set = 0) xend - x.

- **nmax**

This is the maximal number of allowed steps. The default value is 100000.

- **uround**

rounding unit, default 1.0e-16.

- **safe**

The safety factor in step size prediction, default 0.9.

- **facl**
- **facr**

Parameters for step size selection the new step size is chosen subject to the restriction 1/facl <= hnew/hold <= 1/facr. Default values: facl = 5.0, facr = 1.0/8.0

- **nit**

The maximum number of Newton iterations for the solution of the implicit system in each step. The default value is 7.

- **startn**

If startn == 0 the extrapolated collocation solution is taken as starting value for Newton's method. If startn != false zero starting values are used.
The latter is recommended if Newton's method has difficulties with convergence. (This is the case when nstep is larger than naccpt + nrejct; see output parameters). Default is startn = false.

- **npred**

Switch for step size strategy If npred = 1 mod. predictive controller (Gustafsson) If npred = 2 classical step size control
The default value (for npred = 0) is npred = 1.
The choice npred = 1 seems to produce safer results. For simple problems, the choice npred = 2 produces often slightly faster runs.

- **hess**

If hess != 0, the code transforms the Jacobian matrix to Hessenberg form. This is particularly advantageous for large systems with full Jacobian. It does not work for banded Jacobian (mljac < n) and not for implicit systems (imas = 1).

- **fnewt**

Stopping criterion for Newton's method, usually chosen < 1. Smaller values of fnewt make the code slower, but safer. Default min(0.03, sqrt(rtoler))

- **quot1**
- **quot2**

If quot1 < hnew/hold < quot2, then the step size is not changed. This saves, together with a large thet, lu-decompositions and
computing time for large systems. for small systems one may have quot1 = 1.0, quot2 = 1.2, for large full systems quot1 = 0.99,
quot2 = 2.0 might be good. Defaults quot1 = 1.0, quot2 = 1.2.

- **thet**

Decides whether the Jacobian should be recomputed. Increase thet, to 0.1 say, when Jacobian evaluations are costly. for small systems
thet should be smaller (0.001, say). Negative thet forces the code to compute the Jacobian after every accepted step. Default 0.001.

12 SQL support

SQL database is on of the type of low-level storage used by BioUML.

It is possible to work with SQL data via SQL console - special view part for direct SQL manipulations.

SQL based data:

- SQL tables

12.1 SQL console

SQL console allows user to view list of available SQL databases and tables and execute all types of SQL commands. The results of SELECT queries can be shown as table document or exported to separate table data element.

SQL console is realized as separate view part of BioUML.

To connect to SQL database of BioUML server user should specify server address and username/password.

NOTE: In web edition user will login to SQL client automatically with current username/password.

Figure 12.1. Login form of SQL console.

SQL console consists of four parts:

- **query field** (top field) - text input field with SQL query
- **tables field** - list of available tables with database prefixes. Double click on tables automatically generates "SELECT *" query for this table in *query field*.
- **columns field** - list of columns in selected table. Double click on column name automatically adds it to current position of *query field*.
- **history field** - list of last executed commands. Double click on command replaces *query field* with this command.

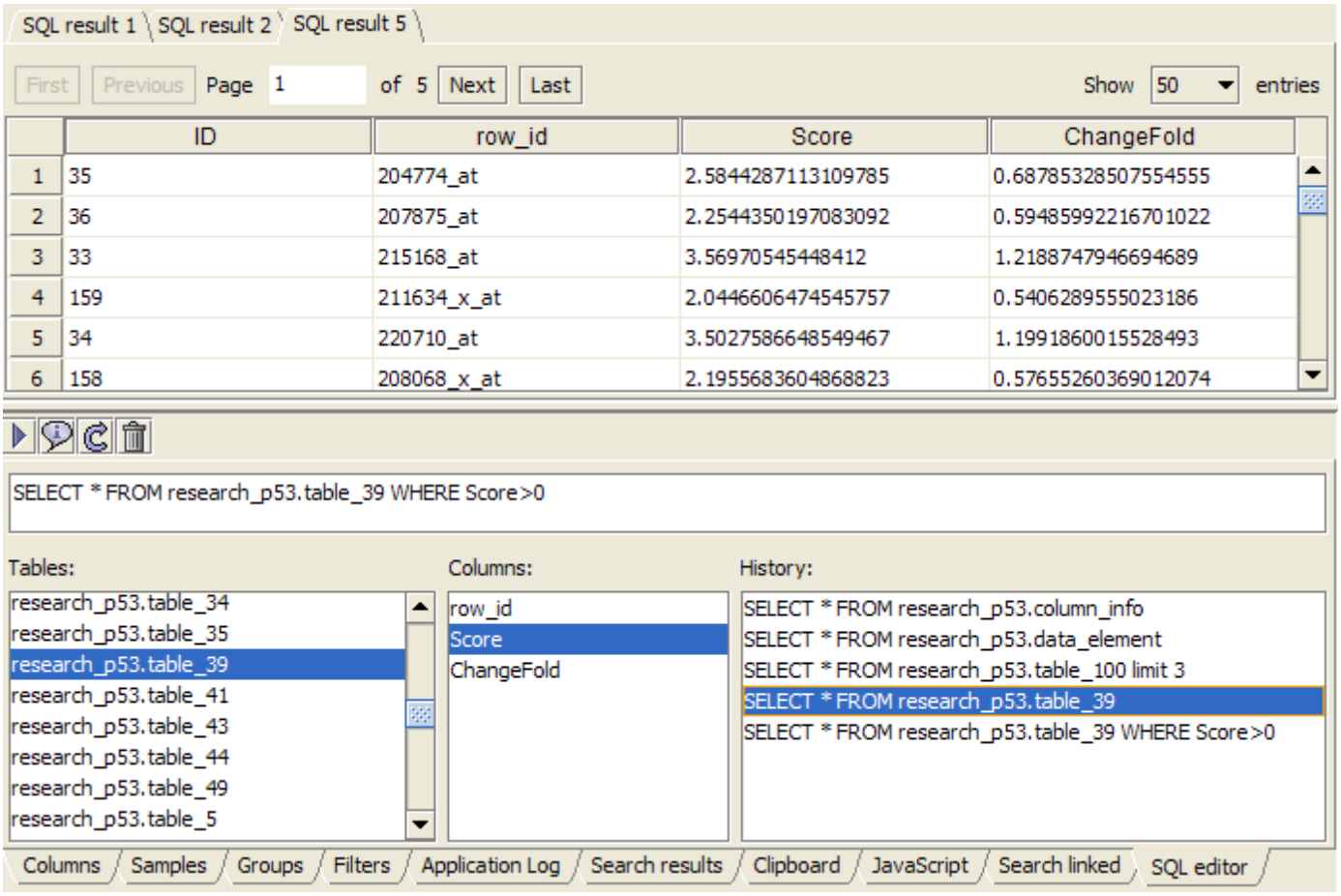


Figure 12.2. SQL console with query examples.

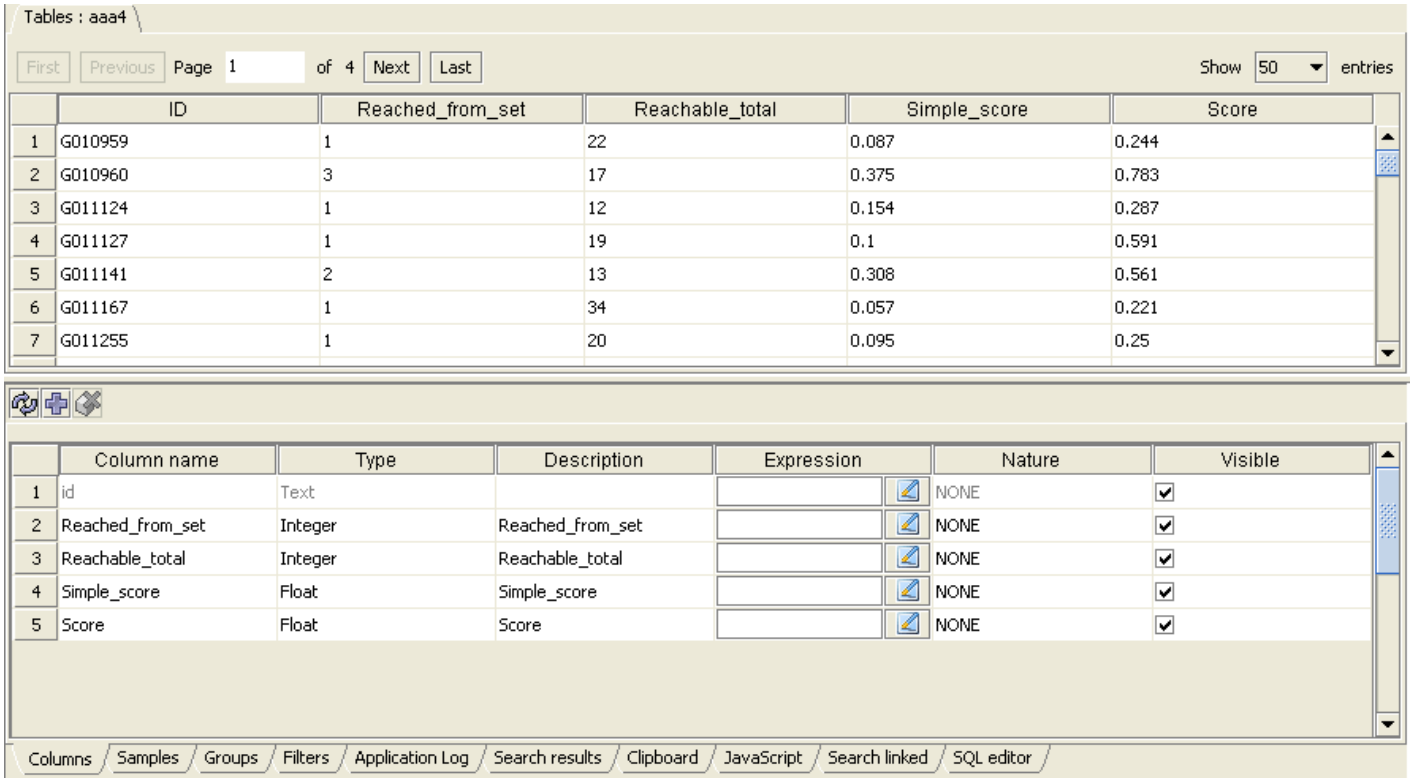
SQL console toolbar

- **Execute query** - execute query from query field and display results as table document if exists.
- **Explain query** - display query explain information as table document. NOTE: explain query is used to optimize difficult SQL queries.
- **Connect to server** - show SQL login form.
- **Clear query field** - remove all from *query field*.

12.2 SQL tables

There are three different types on tables in BioUML: in-memory tables, file-based tables and SQL-based tables. SQL-based tables allows user to work with data directly by using SQL console (for example, standard MySQL command line). All analysis results in research projects by default store in SQL-based tables.

The structure of table in SQL database is the same as structure of table in BioUML.



Tables : aaa4

First | Previous | Page 1 of 4 | Next | Last | Show 50 entries

	ID	Reached_from_set	Reachable_total	Simple_score	Score
1	G010959	1	22	0.087	0.244
2	G010960	3	17	0.375	0.783
3	G011124	1	12	0.154	0.287
4	G011127	1	19	0.1	0.591
5	G011141	2	13	0.308	0.561
6	G011167	1	34	0.057	0.221
7	G011255	1	20	0.095	0.25

	Column name	Type	Description	Expression	Nature	Visible
1	id	Text			NONE	☑
2	Reached_from_set	Integer	Reached_from_set		NONE	☑
3	Reachable_total	Integer	Reachable_total		NONE	☑
4	Simple_score	Float	Simple_score		NONE	☑
5	Score	Float	Score		NONE	☑

Columns | Samples | Groups | Filters | Application Log | Search results | Clipboard | JavaScript | Search linked | SQL editor

Figure 12.3. SQL table example in BioUML workbench.

In this example table *aaa4* is SQL-based table, it's name in SQL database of research project is *table_67*.

```
mysql> select * from table_67 limit 10;
```

row_id	Reached_from_set	Reachable_total	Simple_score	Score
G010959	1	22	0.0869565	0.244436
G010960	3	17	0.375	0.782633
G011124	1	12	0.153846	0.286952
G011127	1	19	0.1	0.590616
G011141	2	13	0.307692	0.561018
G011167	1	34	0.0571429	0.220928
G011255	1	20	0.0952381	0.250267
G011353	1	23	0.0833333	0.590616
G011395	1	9	0.2	0.312771
G011407	1	12	0.153846	0.286952

Figure 12.4. Direct SQL query result for table.

```
mysql> explain table_67;
```

Field	Type	Null	Key	Default	Extra
row_id	varchar(255)	NO	PRI		
Reached_from_set	int(11)	YES		NULL	
Reachable_total	int(11)	YES		NULL	
Simple_score	float	YES		NULL	
Score	float	YES		NULL	

Figure 12.5. Explain result for SQL table.

NOTE: It is possible to use any SQL console (MySQL console in example above). BioUML provides it's own SQL console.

13 Reproducible research

Reproducible research is the common name for functionality for saving, representing and reproducing of the set of data manipulation. BioUML provides special Projects collections and Workflow and research diagrams for this purpose.

13.1 Projects

Project is the group of folders which contains data and history for current research. Before the beginning of new research in BioUML it is recommended to create new project (menu *Research -> New project* in workbench).

Default project structure:

- **Data** - collection with all imported and result data. Data elements can be group by sub-collections (such as Files, Tables, Tracks, etc.).
- **Diagrams** - collection with research and workflow diagrams.
- **Journal** - storage for user activities history.

There are three different types of user activitiies which are saved to the journal:

- Analysis
- JavaScripts
- SQL requests

All of this activities will be stored in journal of current project. Use top toolbar control to select the current project.

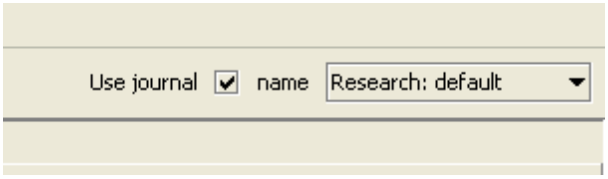


Figure 13.1. Journal selector.

13.2 Workflow and research diagrams

Research diagram

Research diagram allows to represent current research steps(actions) and all input and output data on one diagram. Research diagram can be created by **New project** action in *data/<research_name>/Diagrams* context menu.

There are two basic ways to build research diagram:

- Add action from **History** tab. In this way all input elements for action will be added automatically or linked with exists diagram elements.
- Use top toolbar for creating new elements. In this way building process is the same as typical diagram building in BioUML.

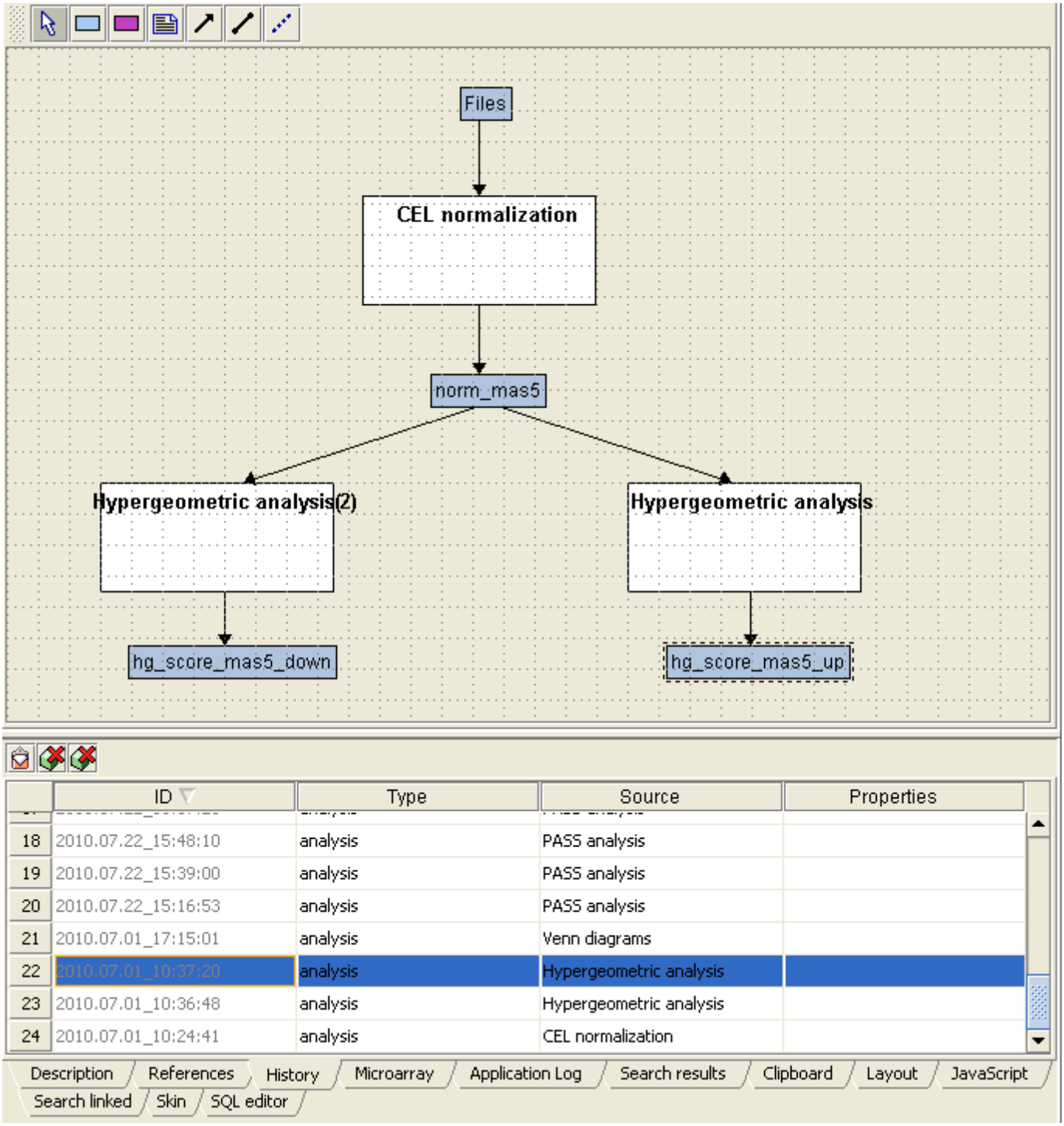


Figure 13.3. Example of research diagram.

Workflow diagram

While the research diagram represents completed research worcflow allows to create scheme for future research and

execute it. Workflow diagram can be created by **New workflow** action in *data/<research_name>/Diagrams* context menu.

The ways to create workflow are the same as the ways to create research. Special **Workflow** tab also allows to set each workflow element properties (such as analysis) and has actions to start or break workflow execution. The status of workflow execution is visible on diagram by elements colors.

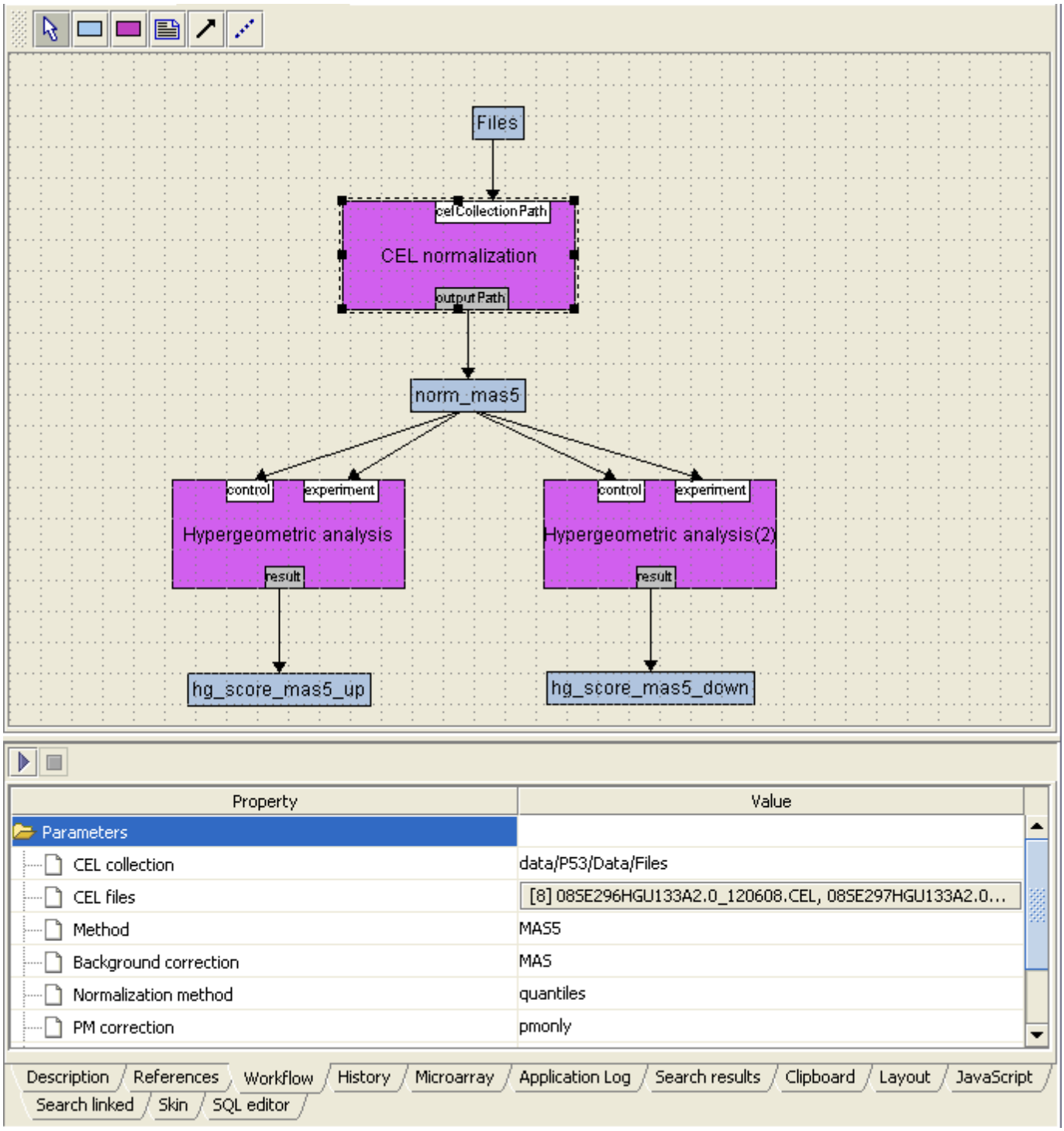


Figure 13.4. Example of workflow diagram.

13.3 Load project dialog

Load project dialog (Figure 13.5) helps a user to configure connection of BioUML workbench with BioUML server for access to remote researches. Setup wizard, step Load research provides the same user interface.

Researches stored on the server in folders data/Examples, data/Protected and data/Public can be installed to corresponding folders of BioUML workbench.

To install remote researches using **Load project dialog**:

1. Select from menu **File > Load project** item. Load research dialog will be opened (Figure 13.5).

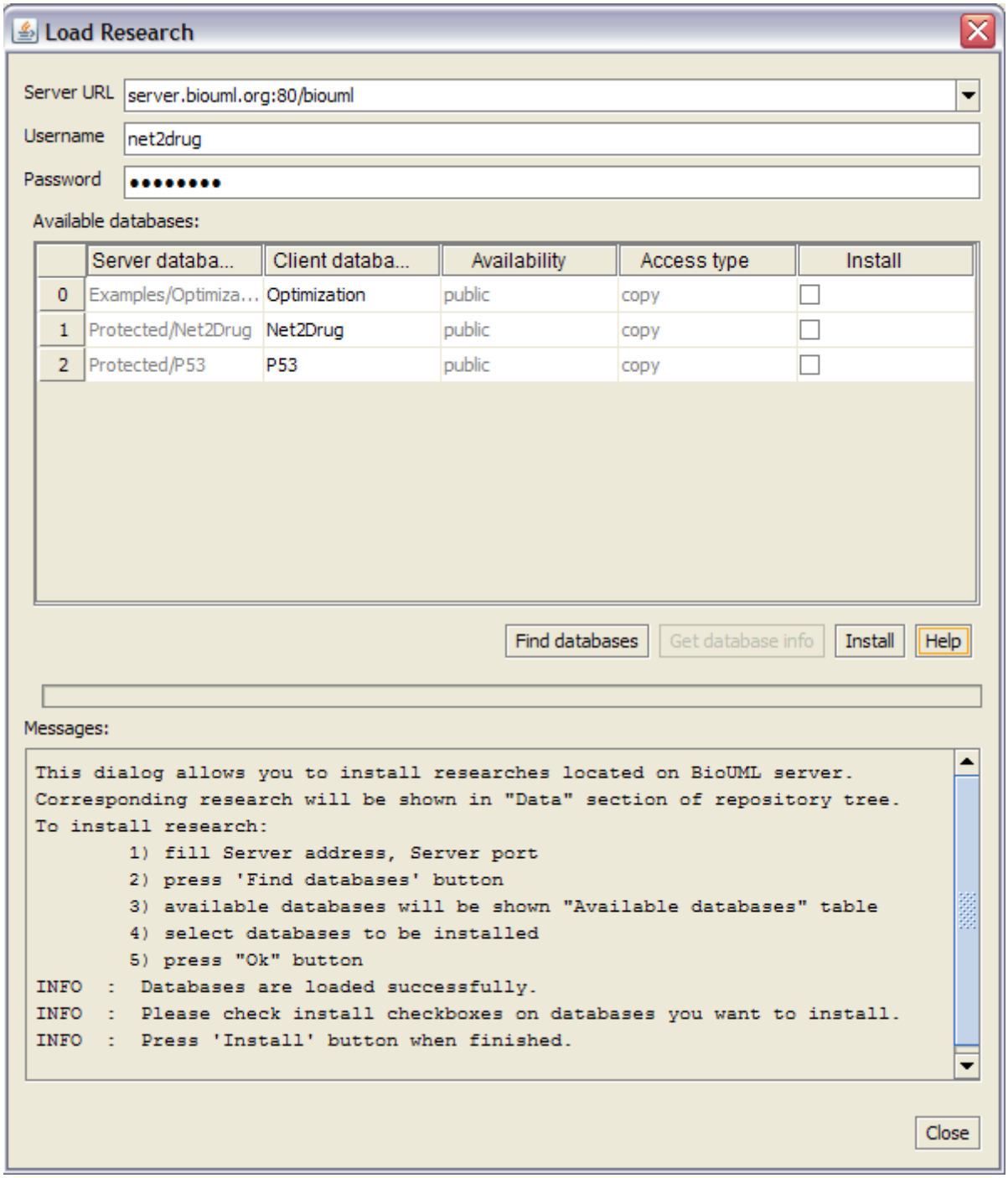


Figure 13.5. Load project dialog.

2. Specify **Server URL** for connection with BioUML server or you can use default BioUML server.

3. Enter **Username** and **Password** for server authorization (use blank values for guest connection)

4. Click **Find databases** button. All researches located on the specified BioUML server will be shown in table **Available databases**.
   The table columns are:

- Server database name - name of the research on BioUML server
- Client database name - name of the database how it will be shown in the repository tree. By default research name on the client side is the same as on the server side.
- Availability - describes the availability of the research. Currently all researches are marked as publicly available.
- Access type - either 'link' (research will be linked and accessed remotely) or 'copy' (research will be copied from the server and accessed locally)

5. To get information about a research:
   - select the research in the table by clicking on corresponding raw;
   - press **Get database info** button;
   - information about the research will be shown in **Messages** pane (Figure 3.3).

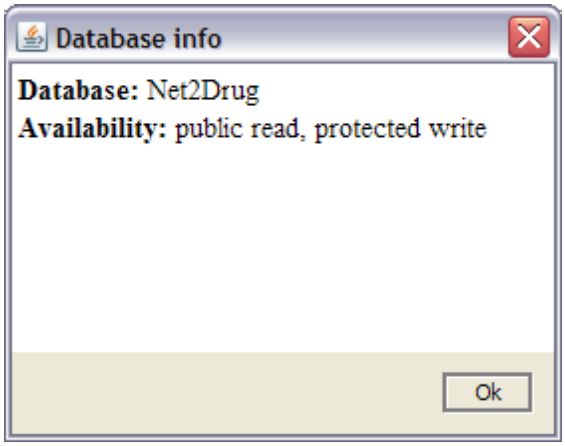


Figure 13.6. Example of research info.

6. Select in **Install** column researches to be installed by clicking on corresponding check box.

7. Press **Install** button. Information about installation process will be shown in **Messages** pane.

8. Press **Close** button to close the dialog after successful installation of remote researches.

Notes:

1. Every research should have unique name. If research with the same name already exists, the research from server will not be copied and warning message will be shown in Messages pane, for example:
   ```
   WARN  :  Research with the same name already exists ('Net2Drug')
   ```

Change the research name in Client database name column, and research will be installed with this name.

See also:

- BioUML server
- Setup wizard

14 To do

There are a lot of chapters and topics to be written. Below is list of main chapters and topics to be written:

- Introduction
  - Main features
- Databases
  - a lot of topics
- User interface
- Simulations
- SBW
- Microarray
- Statistical analyses
- JavaScript
  - functions
- Graphic notation editor
  - user interface
  - JavaScript functions

15 References

Brenner S., cited from R.Bradford, 2009. Salk Signals, 5(2):12-17.

Finney A., 2004. http://sbml.org/wiki/Semantic_Test_Suite

Le Novère N. et al, Moodie S., Sorokin A., Hucka M., Shreiber F., Mi H., Demir E., Wegner K., Aladjem M., imalaratne S., Bergman F.T., Gauges R., Ghazal P., Hideya K., Li L., Matsuoka Y., Villéger A., Calzone L., Courtot M., Dogrusoz U., Freeman T., Funahashi A., Ghosh S., Jouraku K., Kim S., Kolpakov F., Luna A., Sahle S., Watterson S., Goryanin I., Kell D.B., Kohn K., Kitano H., 2009. The Systems Biology Graphical Notation. Nature biotechnology, submitted.

Hucka M. *et al*., 2003. The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models. *Bioinformatics*, **19,** 524-531.

Lee E.A., 2001. Overview of the Ptolemy Project. Technical Memorandum UCB/ERL M01/11, University of California, Berkeley.

Lloyd CM, Halstead MD, Nielsen PF, 2004. CellML: its future, present and past. Lloyd C.M. *et al.,* 2004. *Progress in Biophysics and Molecular Biology*, **85,** 433-450.

Patterson M. and Spiteri R.J., 2003. http://www.netlib.org/ode/odeToJava.tgz

Web sites:

BioModels database - http://www.ebi.ac.uk/biomodels-main

BioPAX - Biological Pathways Exchenge - http://www.biopax.org

CellML - http://www.cellml.org

Eclipse platform - http://www.eclipse.org

Lucene - full-featured text search engine - http://lucene.apache.org/

SBGN - Systems Biology Graphic Notation - http://www.sbgn.org

SBML - Systems Biology Markup Language - http://www.sbml.org

16 Acknowledgements

This work was partially supported by the following grants:

- Volkswagen-Stiftung (I/75941)
- INTAS 03-51-5218
- RFBR 04-04-49826-a
- Siberian Branch of Russian Academy of Sciences (interdisciplinary projects 46, 17, 91)
- European Committee grant 037590 "Net2Drug"
- European Committee grant 202272 "LipidomicNet"