Brandenburg
University of Technology
Cottbus

Faculty of Mathematics,
Natural Sciences and
Computer Science

Institute of Computer Science

COMPUTER SCIENCE REPORTS

Report 02/13

July 2013

# SNOOPY COMPUTATIONAL STEERING FRAMEWORK – USER MANUAL VERSION 1.0

Mostafa Herajy
Monika Heiner

Computer Science Reports
Brandenburg University of Technology Cottbus
ISSN: 1437-7969

Send requests to: BTU Cottbus
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus

Mostafa Herajy, Monika Heiner
snoopy@informatik.tu-cottbus.de
http://dssz.informatik.tu-cottbus.de

# Snoopy Computational Steering Framework – User Manual Version 1.0

Computer Science Reports
02/13
July 2013

Brandenburg University of Technology Cottbus

Faculty of Mathematics, Natural Sciences and Computer Science

Institute of Computer Science

Computer Science Reports
Brandenburg University of Technology Cottbus
Institute of Computer Science

Head of Institute:
Prof. Dr. Ingo Schmitt schmitt@tu-cottbus.de
BTU Cottbus
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus

| Research Groups: | Headed by: |
|---|---|
| Computer Engineering | Prof. Dr. H. Th. Vierhaus |
| Computer Network and Communication Systems | Prof. Dr. H. König |
| Data Structures and Software Dependability | Prof. Dr. M. Heiner |
| Database and Information Systems | Prof. Dr. I. Schmitt |
| Programming Languages and Compiler Construction | Prof. Dr. P. Hofstedt |
| Software and Systems Engineering | Prof. Dr. C. Lewerentz |
| Theoretical Computer Science | Prof. Dr. K. Meer |
| Graphics Systems | Prof. Dr. D. Cunningham |
| Systems | Prof. Dr. R. Kraemer |
| Distributed Systems and Operating Systems | Prof. Dr. J. Nolte |
| Internet-Technology | Prof. Dr. G. Wagner |

CR Subject Classification (1998): I.6.5, I.6.8, D.2.2, J.3

Printing and Binding: BTU Cottbus

ISSN: 1437-7969

# Snoopy Computational Steering Framework
## User Manual
## Version 1.0

Mostafa Herajy and Monika Heiner

– Data Structures and Software Dependability –
Institute of Computer Science
Brandenburg University of Technology
Cottbus, Germany

snoopy@informatik.tu-cottbus.de *

August 21, 2013

*Please sent all questions, comments and suggestions how to improve this material to this address.

# Contents

# 1 Introduction

With the advances of computing power and the proliferation of multi-core processors, it becomes essential to execute long running and computationally expensive simulations at powerful and remote computers – which enjoy high speed computational units – to profit from such precious processing resources. However, such powerful computers do not provide a direct interactive visualisation and analysis of the resulting simulation data due to either the intrinsic patch processing approach of these computers or the sharing of their resources between different users. Thus, there is a need to remotely manage and analyse the simulation output traces simultaneously, while the simulation is still in progress. Correspondingly, many different techniques have been proposed to overcome these limitations. Computational steering is among the elegant and promising tools that provide a tight coupling between simulation and visualisation modules for scientific computational models.

In this manual we discuss the use of Snoopy's computational steering framework to simulate and interactively steer (stochastic, continuous, hybrid) Petri nets, e.g., biochemical network models. There are many important functionalities that are provided by this framework in order to facilitate the conduction of "wet-lab" experiments. If you are wondering about what these tools can add to your work-flow, it might be worth reading the following paragraphs.

Here are some aspects of what you can do using the Snoopy steering framework:

- **Remotely run and control a simulation**

  You can run your simulation at a remote computer. This feature allows the control of high computational power machines from a local, typically less powerful computer. The simulation will run at a server machine, while the visualization of the results is done at a different machine serving as GUI client.

- **Execute the same model using different simulation algorithms**

  Sometimes it is useful to study a model in different paradigms, i.e., stochastic, continuous, or hybrid. In this framework, the same model definition can be executed with different simulation algorithms.

- **Manage concurrently different models with possibly different simulators**

  Different models can be simultaneously executed at the server side. Each model is assigned a separate simulator and is executed independently from other simultaneously running models.

- **Define different result views to explore the simulation results**

  Result views (or views for short) provide a quick means to explore model results from different perspectives. Each view is defined by a set of curves and their associated attributes. Different views can be defined for the same model.

- **Explore on the fly your running models**

  Using the steering graphical user interface (Steering GUI), you can easily navigate among different models. Besides, the list of running models at the server side can be refreshed if another user adds a new model to the server.

- **Steer the simulation parameters while the simulation is running**

  The main goal of Snoopy steering framework is to enable users to interact with their models during simulations. Users can change model parameters as well as the current marking and immediately monitor the system's response to such changes. This is an useful tool since a user is allowed to ask "what-if" questions.

- **Control the simulation speed**

  The simulation speed can be set to an appropriate level to facilitate the interaction with a running model during its execution. This feature is important if the simulation parameters are allowed to be changed while the simulation is running.

- **Connect to your simulation at any time from whatever place**

  The overall organization of this framework is flexible to let users connect/disconnect to/from running models without affecting their execution. Moreover, you can connect to your models from different places, for example from your office or from your home.

- **Collaborate with other people while executing model dynamics**

  More than one user is permitted to connect to the same models. Users can collaborate in executing and steering a running simulation. This feature might promote the sharing of knowledge between different users with different backgrounds.

- **Platform-independent implementation**

  The core communication library is written in standard C++, and therefore it can run on different platforms among them are windows, Mac OS X, and Linux. Moreover, clients and servers do not need to run under the same platform.

# 2 Overview

In this section we give a general overview of the Snoopy steering framework to help understanding the high level organization before going into details.

## 2.1 Framework

Figure 1 presents the general architecture of the Snoopy steering framework. Its main components are: the steering server, the steering graphical user interface, the steering application programming interface (APIs), and the internal and external simulators. These interdependent ingredients enable users not only to run their biochemical network models and obtain results, but also to share, distribute and interactively steer them. Additionally, users do not need to wait until the simulation ends in order to discover potentially incorrect results. Instead, using this framework, errors could be discovered early and be immediately corrected during the simulation and, if necessary, the simulation could be restarted using the current setting. Subsequently, the overall time required to carry out dry-lab experiments will substantially decrease.

The main component of the architecture is the steering server. It is the central manager of the model data and communication traffic between the different framework components. It is a multi-user, multi-model, multi-simulator, and multi-threaded server. Inside the server, data is organised in terms of individual models which are defined by means of Petri nets.

The steering graphical user interface is the user's entry point to the overall architecture. Through it, the user can monitor and steer the simulation output and the corresponding key parameters, respectively. Users can flexibly connect and disconnect from their local machines to the available steering servers and view the currently running models. Model dynamics are produced using either an internal or an external simulator. Internal simulators are implemented inside the server which currently supports deterministic, stochastic, and hybrid algorithms, while external simulators are defined by the user and dynamically linked to the running server.

The steering application programming interfaces (APIs) are used to incorporate external simulators into the steering server. Additional responsibility of the API library is to facilitate the connections between the different framework components. More specifically, it is used to carry out the communication between the steering GUI and the steering server.

Finally, this framework permits the simulation to be remotely executed using an external simulator developed by the user (optional component). The communication

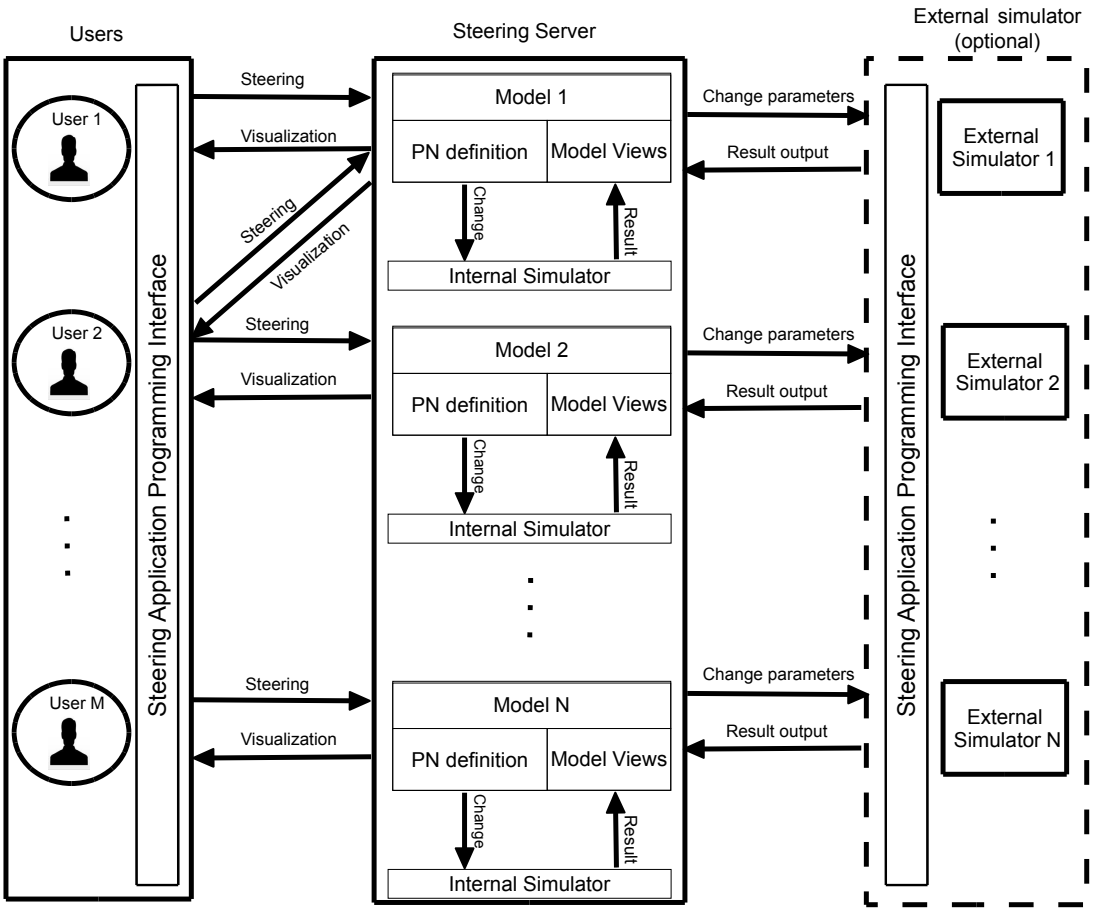



Figure 1: Petri nets and computational steering framework. The framework consists of four components: steering server, steering graphical user interface (GUI), steering application programming interface (Steering API), and simulators (internal and external). The flow of information goes in two opposite directions: from the simulator to the user (monitoring) and from the user to the simulator (steering). The data structure inside the server is organised in terms of Petri nets: places, transitions, arcs and parameters. The place data structure includes the initial marking, and the transition data structure the rate functions. A model can contain different result views, which are defined by the users and submitted to the server for further download and manipulation.

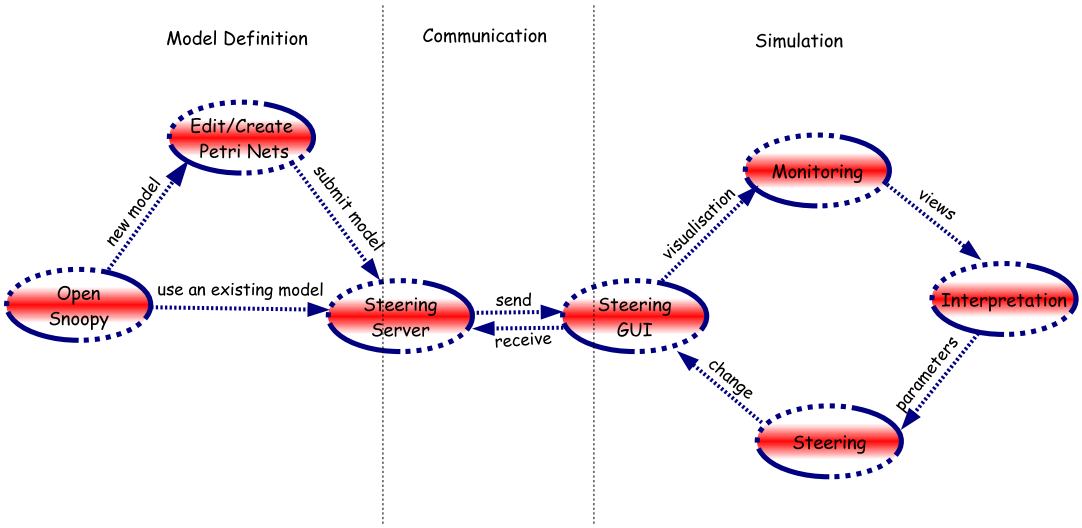



Figure 2: Graphical illustration of a typical application scenario of Snoopy's steering framework. The user has two options at the beginning: either reading one of the models already existing in the server or submitting a new model. In the latter case the Petri net model can be created using Snoopy or other Petri net editing tools by use of our API library. Afterwards, Snoopy's steering GUI can be used to perform the monitoring and steering.

between these external simulation modules and the other architecture components takes place through the steering APIs. This means that with modest effort, users can include their own favourite simulators and perform the monitoring and steering tasks by help of the other framework components.

## 2.2 Application Scenario

In a typical application scenario, a user constructs the biochemical reaction network using a Petri net editing tool (e.g., Snoopy). Afterwards, the (stochastic, continuous, hybrid) Petri net model is submitted to one of the running servers to quantitatively simulate it. Later, other users can connect to this model by their steering GUIs. One of the connected users initialises the simulation while other users could stop, pause, or restart it. When the simulator initially starts, it uses the current model settings to run the simulation. Later, other users can remotely join the simulation and change model parameters or the current marking. Figure 2 illustrates graphically a typical application scenario of Snoopy's computational steering framework.

Table 1: Minimal and Optimal Hardware Requirements.[0]

| Requirements | **Snoopy** | | **SSServer** | |
|---|---|---|---|---|
| | Minimal | Optimal | Minimal | Optimal |
| Processor | 1 GHz | 2 GHz | 2 GHz | 2.5 GHz or higher |
| RAM | 256 M | 1 GB | 512 M | $\geq$ 8 GB |
| Free Hard Disk Space | 500 M | 2 GB | 500 M | 10 GB |
| LAN adapter | X | X | X | X |

# 3 System Requirements

In this section we outline the hardware and software requirements to run Snoopy and the Snoopy steering Server (SSServer) on your computer.

## 3.1 Hardware Requirements

The hardware requirements depend on your specific needs. For instance, if you plan to run big models (100,000 to 1000,000 variables), then higher requirements are needed than to run relatively small ones. At the time of writing this manual, a 64Bit machine, preferably running native Linux is recommended for computational expensive models. Table 1 outlines the minimum and the optimal hardware required to run the Snoopy steering framework.

## 3.2 Software Requirements

The Snoopy steering framework implementation is platform-independent. Therefore you can use it on your favourite operating system. In more specific terms, you will need one of the following operating systems running on your computer:

- Window XP or higher
- Mac OS X 10.5 or higher
- Linux, e.g., Ubuntu

[0]This table is both for 32 and 64 Bit machines.

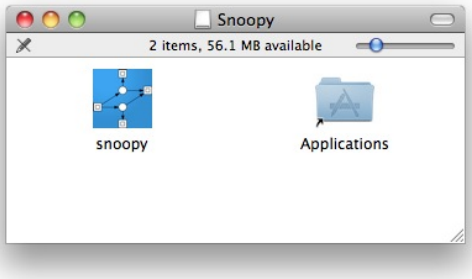



Figure 3: Snoopy Setup

Next, you will need a copy of Snoopy suitable for your operating system which can be downloaded from `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy`.

Finally, a copy of the SSServer is required to run this framework. Please note, at the time of writing this manual, the SSServer is not available at Snoopy's website. However, it can be requested from the authors by email.

# 4 Installation

The specific installation procedure depend on your operating system version. Below we give two examples using Windows and Mac OS X.

## 4.1 Install Snoopy under Mac OS X

To install Snoopy under Mac OS X, first acquire a Snoopy version. The Snoopy setup file under Mac OS is in a disk image format (dmg). All the necessary data are provided in a single file. To install this file on your computer do the following steps:

- Mount the disk image on your computer by double-clicking the dmg file.
- The disk image will appear as another CD in your Finder with the name "Snoopy".
- Opening this disk, you will find the Snoopy application bundle, as shown in Figure 3.
- Copy Snoopy to your desired location, or drag and drop it into the application folder.

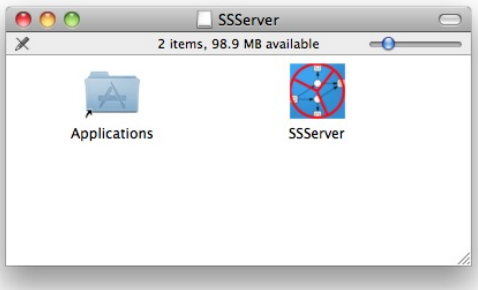



Figure 4: SSServer Setup

## 4.2 Install SSServer under Mac OS X

To install the SSServer to your Mac OSX, similar steps as for installing Snoopy are required. We repeat them again for your convenience.

- Mount the disk image on your computer by double-clicking the dmg file.
- The disk image will appear as another CD in your Finder with the name "SSServer".
- Opening this disk, you will find the SSServer application bundle, as shown in Figure 4.
- Copy the SSServer to your desired location, or drag and drop it into the application folder.

## 4.3 Install Snoopy under Windows

To install Snoopy under Windows, follow these steps:

- Obtain the windows installer package for Snoopy. This should be a msi file.
- Double click the obtained setup package.
- The windows installer will start as shown in Figure 5.
- Follow the simple instructions in this window by hitting the next button.
- At the end of these steps Snoopy should be installed on your computer and a shortcut will be created at your desktop.
- To start Snoopy double click the Snoopy's shortcut from your desktop.

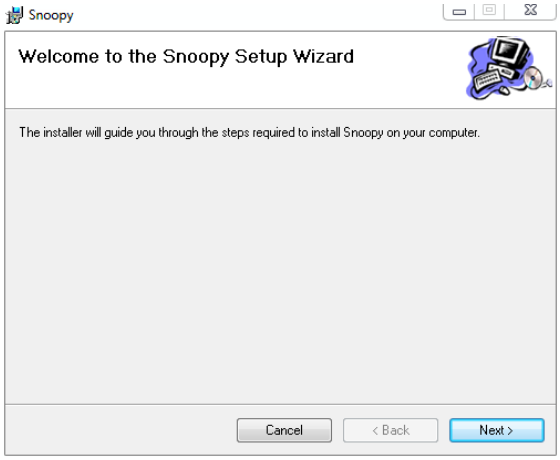



Figure 5: Snoopy Setup under Windows

### 4.4 Install SSServer under Windows

Similar steps are required to install the SSServer under Windows. For your convenience we repeat them in the following procedure.

- Obtain the windows installer package for SSServer. This should be a msi file.
- Double click the obtained setup package.
- The windows installer will start as shown in Figure 6.
- Follow the simple instructions in this window by hitting the next button.
- At the end of these steps SSServer should be installed on your computer and a shortcut will be created at your desktop.
- To start the SSServer, double click the SSServer's shortcut from your desktop.

## 5 Getting Started

### 5.1 Launching Snoopy and SSServer

The first step to use Snoopy in the steering mode is to open the Snoopy Steering Server (SSServer) on your local computer or on another remote machine where you want to run the simulation. Second you need to open Snoopy on your machine. The exact procedure


Figure 6: SSServer Setup under Windows

depends on the operating system you use. We assume that you have basic knowledge to run software on your operating system.

After starting the server, it will look as shown in Figure 7 under the Mac OS X. Other platforms might have slightly different, but similar interfaces. There are two important information you need to notice after starting the SSServer: the running server name (IP) and the listening service number. They are written in the log window after the SSServer started successfully. You will need these information when connecting to a server using Snoopy. For instance, in Figure 7, the server name is swqlab3.informatik.tu-cottbus.de and the IP is: 141.43.202.57. The server is listening to the service number 3000.

In the status bar, there are some useful information about the status of the SSServer. The first one gives information about the current state of the server. For example in Figure 7, the SSServer state is "Running". Other possible states are "stopped" and "Restarted". Make sure before connecting to a server that its state is "Running". The second information in the status bar is the number of currently running models inside the server. Each time you submit a new model, this number will increase. The third and final information is the number of clients (users) that are currently connected to the server.

After you have started the SSServer, you need to open Snoopy on your machine. Figure 8 gives a screenshot of Snoopy under Mac OS X.

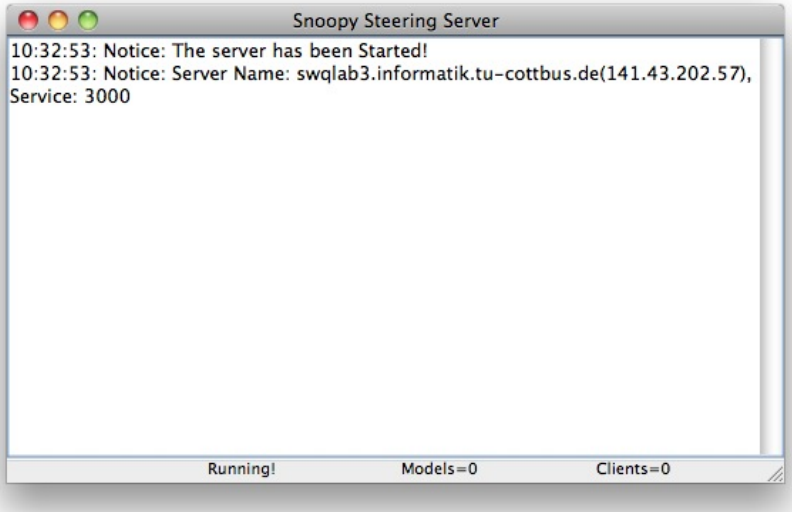



Figure 7: Snoopy Steering Server under Mac OS X

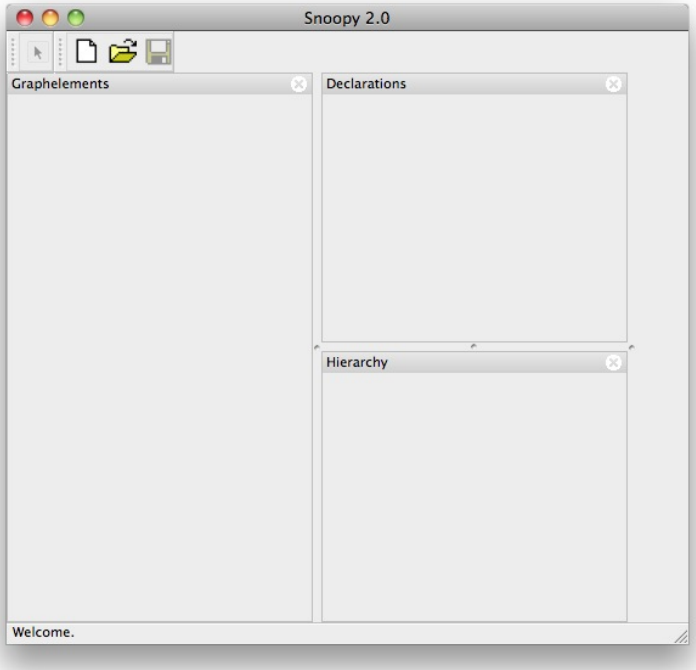



Figure 8: Snoopy under Mac OS X


Figure 9: Open Snoopy in the steering mode (here: with no Petri net open)

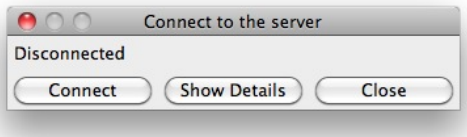



Figure 10: Connection window

## 5.2 Connecting to the Server

To connect to a running SSServer using Snoopy, you first need to decide if you would like to use a model from those running on the SSServer (if there are some) or if you would like to submit a new model. In the latter case you will need to open your Petri net model using Snoopy. Nevertheless, to open the connection window, do the following steps in either of these cases:

1. From the view window, select "Start Steering-Mode" (see Figure 9) or just press "F7" from the function keys if you prefer using short cut keys.

2. The connection window will be opened (see Figure 10).

After you have opened the connection window you can connect to the sever using the default setting via the "connect" button. This makes sense if you run the SSServer at the same computer as you use Snoopy or if you already configured Snoopy to connect to a certain SSServer. However, if you run the SSServer on another computer, you must first configure the connection before pressing the "connect" button, see Section 5.1.

Another important point is that the SSServer and Snoopy should have the same version number of the communication library. You cannot let Snoopy connect with an SSServer having a different version. A check will be done at the beginning and you will be informed if an incompatible version is detected. In this case the Snoopy and SSServer versions are displayed for your convenience.

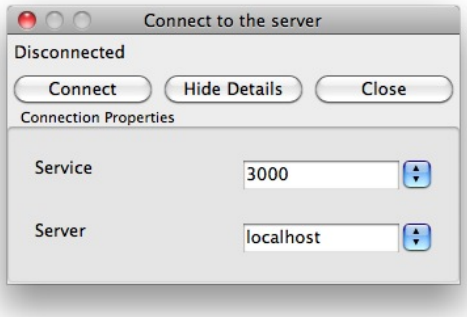



Figure 11: Configuring connection setting

### 5.2.1 Configuring Connection Settings

To configure the connection settings to connect to a certain SSServer do the following steps:

- Press the "Show details" button from the connection window.
- New fields appear as illustrated in Figure 11.
- In the service text box, enter the service number as discussed in Section 5.1.
- In the server text box, enter the server name or the IP address.

The default value of the service number is "3000", and the default for the server name is "localhost". To simplify entering these information, you can also select from previous setting using a list.

### 5.2.2 Saving Connection Settings

The connection settings, which you have entered in the previous step, are automatically saved to the Snoopy registry. Therefore you do not need to re-enter them again when you later want to re-connect to the same SSServer.

## 5.3 Submit a New Model

To use a new model in the steering mode do the following steps:

1. Open the Petri net file in Snoopy.

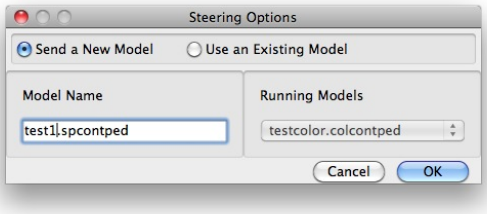



Figure 12: The steering options window

2. Open the connection window as discussed in Section 5.2.
3. Make sure that you correctly configured the connection settings to connect to a running server.
4. Hit the "Connect" button. The steering option window will appear as illustrated in Figure 12.
5. Make sure, the "send a new model" option is selected.
6. Enter a model name and then press ok.

Please note that the name of the loaded file in Snoopy is used as a default name for the new model. You can edit the name as you want under the condition that there is no model running on the SSServer side with the same name.

After pressing the Ok button, the steering dialogue will open as shown in Figure 14.

Please note, if you use a coloured model, then the unfolding window will automatically appear after step 6, because you need to unfold the model before submitting it to the server. To unfold a model, press the "Start" button on the unfolding window as shown in Figure 13.

## 5.4 Using an Existing Model

The Snoopy steering framework gives you the chance to use existing models that are running on the SSServer without opening them in Snoopy. For example, let us assume that you have one model that is already running in the SSServer and you want to disconnect and return back to it later. In this case, you do not need to reopen the Snoopy file or even to unfold it if it is a colored model. To steer an existing model, follow this procedure:

1. Open Snoopy, and then open the connection window as described in Section 5.2.

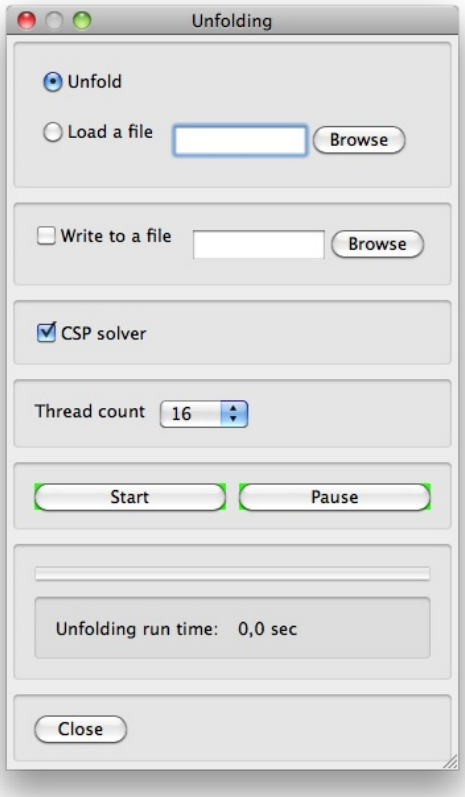



Figure 13: Automatic unfolding window

2. Press the "Connect" button.

3. A new window will appear with the models currently running on the SSServer.

4. Select one of these models and press ok.

5. The steering dialogue will open with the selected model loaded.

Please note, if there are no running models on the server side, then you will be informed and the process of opening the steering GUI will terminate. Besides, you can do these steps while a Petri net model is opened in Snoopy. However, you will need to select the option "Use an Existing Model". After that you can select one of the running models from the list.

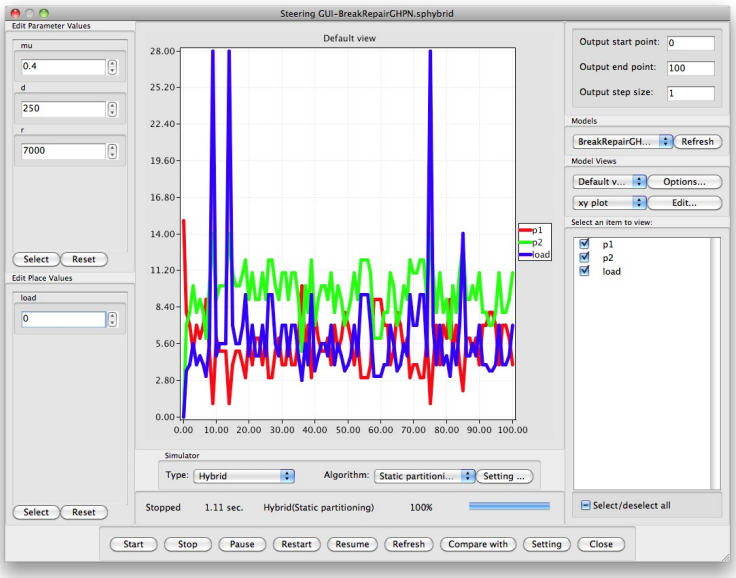



Figure 14: The steering GUI

### 5.5 Replacing an Existing Model

When constructing a Petri model for the first time, you might need to submit the same model several times with some modifications. Such modifications might involve adding new places or new transitions to the model definition. In this case, submitting the same model with a different name will increase the number of models running on the SSServer. A better procedure is to submit the modified model with the same name as the old one. In this case the modified model will replace the previous one. Please note, in order to replace an existing model, there should be no other users connected to it. If there are other users connected, the server will refuse to replace this model, and the client will get a notice.

## 6 Steering GUI

The ultimate goal of the Steering GUI is to provide the user with a remote control-like facility to interact with the currently running models. The connection between the steering GUI and the steering server is dynamically established, meaning that a connection does not need to be set up in advance before the simulation starts.

Among the helpful features that Snoopy's steering GUI provides are: viewing the models running on a remote server, selecting between different simulator algorithms,


Figure 15: Model list

changing the simulation key parameters and the current Petri net marking, providing the user with different views of the simulation results including intermediate and final outputs, and remotely changing the simulator properties. Figure 14 provides a screenshot of the steering GUI.

## 6.1 Models

Through the steering GUI you can explore the currently running models. A model refers to the Petri net definition, supplemented by a set of metadata associated with it (see the next subsection for a more precise description). To view the models currently running on the server side, you will find a list box in the right panel of the steering GUI. Drop down this box, a list of models will be displayed as illustrated in Figure 15.

### 6.1.1 Model Skeleton

A model in the Snoopy steering framework is mainly defined in terms of Petri nets. More precisely a model can be defined by specifying the following items:

- **Model Definition** This includes the Petri net used to define the problem under consideration which in turn is defined by: a finite non-empty set of places, a finite non-empty set of transitions, and the connections between the places, transitions, rate functions, and initial marking. We assume here that the reader is familiar with the use of Petri nets. For more information about using Petri nets, see, e.g., [HH12a] from the reading list.
- **Model Views** Each model is associated with a set of data structures called (result) views. Views are an elegant and quick means to explore the model's result from different perspectives. For more information about defining and using views see Section 6.2.
- **Metadata** Models contain additional information about how it can be executed. A few examples of these information are: the name and type of the simulator used to execute the model, simulator settings, . . . , etc.

### 6.1.2 Types of Model

There are two main types of models when using the Snoopy steering framework:

- **Uncoloured models**

  Uncoloured models are the standard low-level Petri nets without additional coloured information. They do not require unfolding in order to use them in the steering mode.

- **Coloured models**

  Uncoloured models do not scale easily, particularly if we need to define models with repetitive components. Therefore coloured models use colours to let a user define bigger models in a concise manner. Coloured models, however, require an (automatic) unfolding before they can be used in the steering mode.

### 6.1.3 Refreshing the Model List

The list of models appearing in the list box are only those which did exist when the steering GUI client was connected to the SSServer. These models may change if someone else adds/removes a model. To refresh the models in the list box, hit the refresh button next to the model list box. The list of models will be updated.

### 6.1.4 Changing the Current Model

To change the current model, which is used in the steering GUI, and use another one from those running on the server side, follow these steps:

- Go to the model list box on the right side of the steering GUI.
- Drop down the model box, a list of running models will be displayed.
- Select a model from the list.
- A message box will be displayed asking you if you would like to save your local changes to the server.
- The model information will be downloaded to your client. This might take some time depending on the model size.

Please note, saving your local changes to the server side will overwrite the model settings there and, thus, they will be available for other users. For instance, if you

defined some result views (see next section) using your local client and then you opt to save your changes to the server, a user B which connects to this model will be able to use your views.

## 6.2 Result Views

In this section we discuss a model component called result view, or view for short.

### 6.2.1 What Are Views

Models could contain multiple views to provide more elaborated ways of understanding the simulation results. Result views are defined by the user via the steering GUI. They are saved on the server side and can further be downloaded whenever a user opens a model. A view is defined by selecting a subset of the model's places or transitions to monitor their values during the simulation. Each selected place (transition) gets assigned a curve inside the view. A curve is defined by a set of attributes, e.g., colour, style, width and so on. In Snoopy, when a model is executed for the first time, a default view is created.

Besides, each view is associated with a viewer. The viewer determines how the view's data is displayed. For instance, a tabular viewer displays the simulation output using a table.

Views come along with many advantages to increase the user's experience of using the steering framework. They provide an easy way of exploring the model results. Having several views defined for a model, a user can explore the results by just switching between these views. Additionally, views increase the collaboration among users. Users at different computers are able to define different views independently from each other. After that they can share them by submitting their views to the server. Finally, views can be defined "on the fly" while the simulation is running. Users do not need to interrupt the simulator to define a new view.

The currently defined views can be located at the right side of the steering GUI window. There, you will find a list box that contains your local views, i.e., the user copy of the result views, holding any changes before committing the changes to the server.

### 6.2.2 Default View

When you run a model for the first time using the Snoopy steering framework, a default view will be created. This new view will obtain the name "Default View" and it will

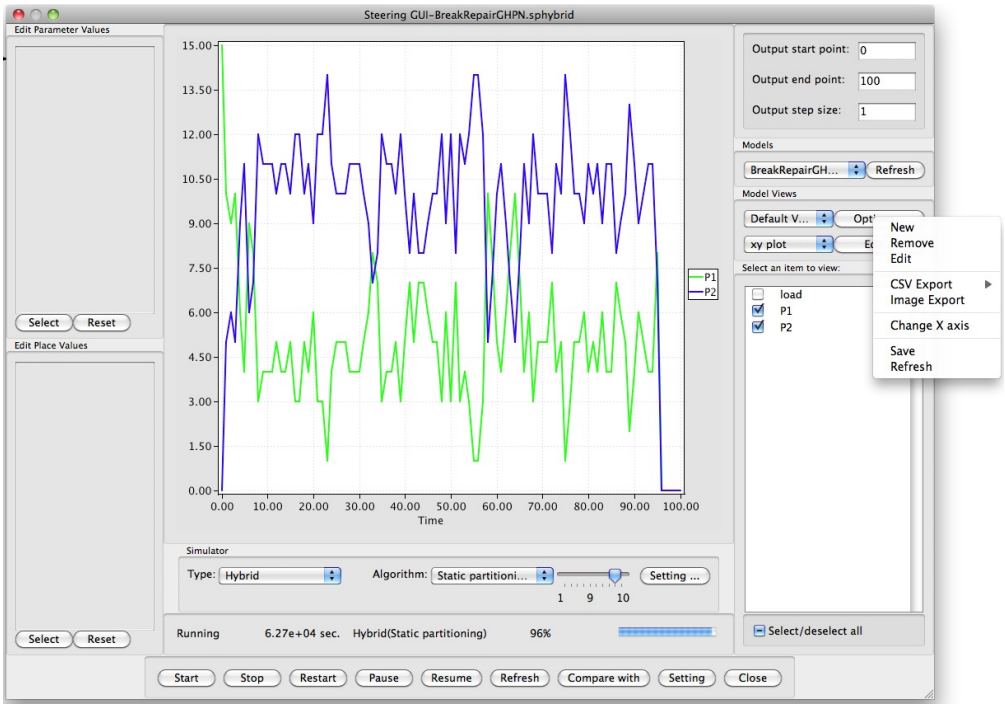



Figure 16: Creating a new view

contain all the (coloured) places of the net as default curves. If you load an existing Snoopy file, then all views which are defined inside this file will be loaded by default.

### 6.2.3 Create a New View

You can create as much views as you want. There is no limit for the number of views that can be created for a single model (unless your computer resources do not allow that).

To create a new view, follow these steps:

- Go to the "Option ..." button next to the view list box.
- Left click the "Option ..." button, a pop-up menu will be displayed as illustrated in Figure 16.
- Select "New" from the menu. A new dialog will open asking you to enter a name for the new view, compare Figure 17.
- Give a name to the view and press Ok.

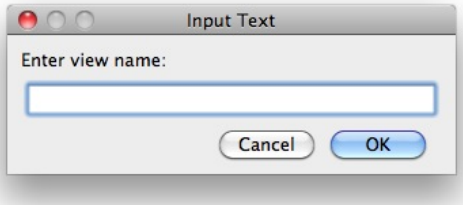



Figure 17: Enter view name dialog

- The new view will be added to the list.

Please note that view names should be unique. That is you cannot assign the same name to different views. Moreover, a view name should not be empty. Besides, when a new view is created, the setting of this view will be copied from the current selected one. Therefore you might not notice any changes to the steering GUI having created a new view, except the name of the view. In the following you will learn how to adjust the view's settings.

### 6.2.4 Navigate to a View

By default, when you create a new view, this view will be the current one. To change the current view and select another one, do the following steps:

- Go to the views' list box in the views' panel.
- Left click the list box, a list of views will be displayed.
- Select one of them and hit the mouse button.
- The selected view setting will be loaded in the Steering GUI.

### 6.2.5 Edit a View

After you have defined a new view, you can edit some properties of it (e.g., the curve list, viewer, and so on). In this section we focus on editing the number of curves associated with a view. To edit a view, do the following steps:

- Select the view that you want to edit as the current one.
- Left click the "Option ..." button.

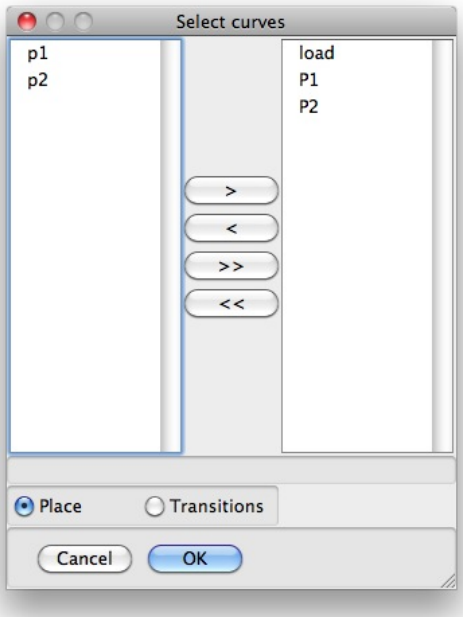



Figure 18: Edit view dialog

- From the pop-up menu select "Edit".
- The "select curves" dialogue will open as illustrated in Figure 18.
- Select the node type (place or transition), the corresponding list of nodes will be displayed on the left list box (available nodes).
- You can use the buttons on the middle dialogue panel to move nodes from the available list to the selected one.
- Having selected some nodes, hit the "Ok" button.
- The selected nodes will be displayed on the item list panel.

Please note, when you change the node type (i.e., place to transition or vice versa), the list of selected nodes will be cleared. This means that you cannot mix places and transitions. Moreover, the dialogue in Figure 18 will adapt itself based on the Petri net type (coloured/uncoloured). Therefore, if you open the dialogue while a coloured Petri net is loaded, then the dialogue will look as in Figure 19 where you will have the option to select either coloured or uncoloured nodes.

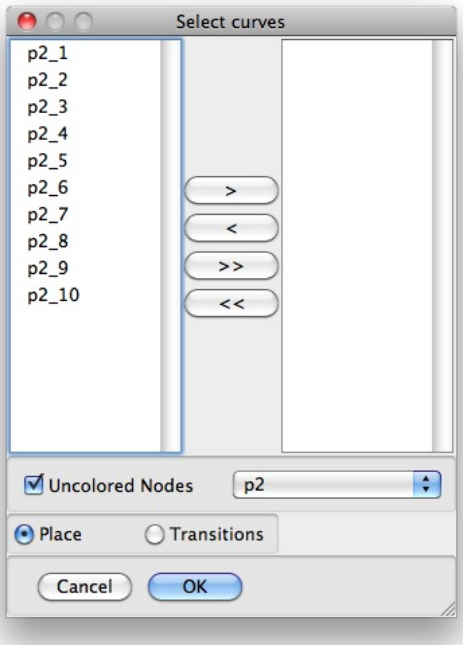



Figure 19: Edit a view of a coloured model

### 6.2.6 Delete a View

To delete a view from the currently defined views do the following:

- Navigate to the view that you want to delete and select it as the current view.
- Left click the "Option ..." button.
- Select "Remove" from the pop-up menu.
- Confirm the warning message.
- The current view will be deleted and the previous or the next one, respectively, will be set as the current view.

Please note that you will not be able to delete a view if there is only one defined for a model.

### 6.2.7 Change Curves Properties

Each view contains a number of curves. These curves represent places or transitions. You can change the attributes associated to each curve, e.g., colour, line width and line

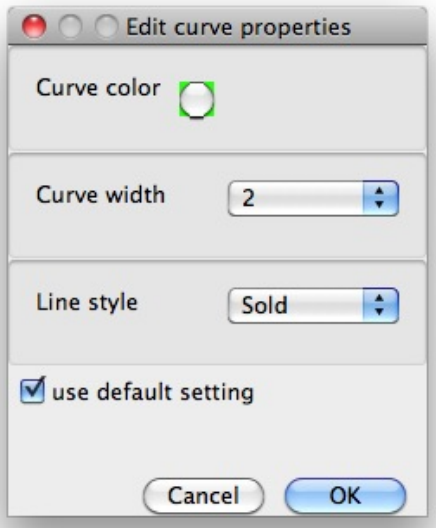



Figure 20: Edit curve properties

style. If you do not want to change these attributes individually and you would like to make a change to all of them, then see Section 6.3.3.

To change the attributes of a curve do:

- In the list of the item list box double click the curve of which you want to change its properties.
- A dialog will be displayed as in Figure 20.
- Change the values of some properties and press Ok.
- If you want to use the default settings, you can check the box "use default setting".
- The curve properties will be changed in the steering GUI.

### 6.2.8 Export a View

A view can be exported as a comma separated values (CSV), or as image format.

**Export to CSV** To export the view to a CSV file do the following:

- Left click the "Option ..." button.
- From the pop-up menu select "CSV export".
- From the sub-menu you can export either the currently selected items or all of the places/transitions.
- A new dialogue will open. From this dialogue you select a file name and the separators.
- Click export.

Depending on your purpose of using the CSV file, you can select a suitable separator. The separator can be a tabular, comma, semicolon, or colon.

**Export to an image format** You can export the current view to a number of image formats, e.g., bmp, gif. To export the current view to an image file do the following:

- Left click the "Option ..." button.
- From the pop-up menu select "Image export".
- A dialogue will open as illustrated in Figure 21.
- Select a name for the image file as well as the path.
- Next to "Format", select the image format.
- Hit save.

Please note, exporting a view to a postscript format is currently not available.

### 6.2.9 Save Model Views

Having made modifications to the views that are associated to a model, you can submit such changes to the server side where other users can further download them or you can use them when you reconnect to the server. Saving result views involves saving all the local changes related to them. To save views, do the following steps:

- Left click the "Option ..." button.
- From the pop-up menu select "Save".

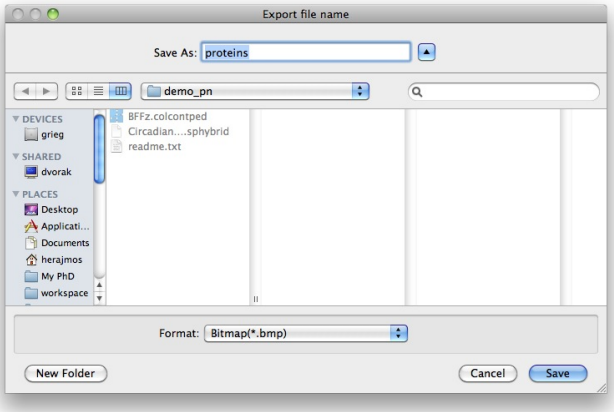



Figure 21: Exporting the view to an image format

- Confirm the warning message.

Please note that this action will overwrite any changes related to this result view on the server side including those that were done by other users.

### 6.2.10 Refresh Model Views

If another user makes some changes to the result views (e.g., creating a new view), and you would like to use these settings instead of redoing them again, you can use the refresh tool. To refresh model views do:

- Left click the "Option ..." button.
- From the pop-up menu select "Refresh".
- Confirm the warning message.

## 6.3 Viewers

In order to visualise the simulation results you need to select one of the available viewers. Viewers are associated with a view. That is each view is assigned a viewer to show the simulation output. Latter, when you change the current view, the viewer is changed too.

### 6.3.1 Types of Viewer

Currently, Snoopy supports three types of viewers: tabular (see Figure 22), XY-plot (see Figure 16), and histogram plot (see Figure 23). A tabular viewer shows the data in a

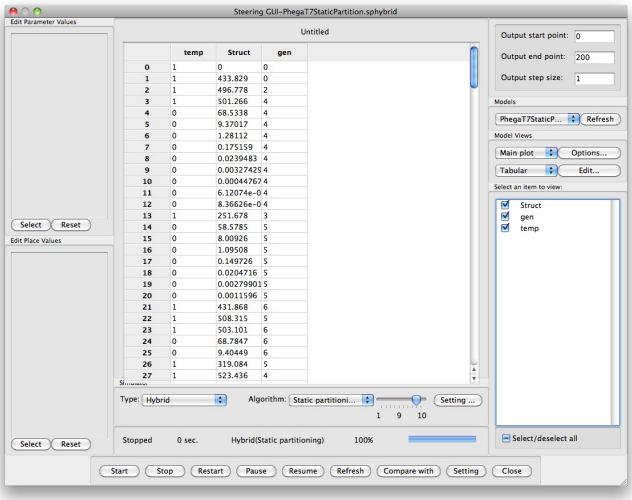



Figure 22: Example of tabular viewer

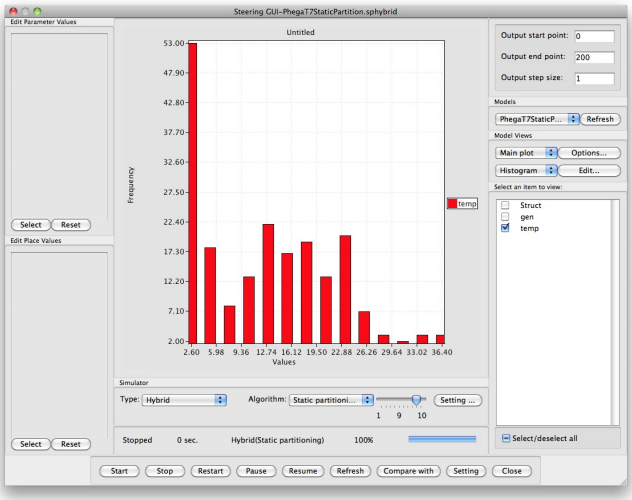



Figure 23: Example of histogram plot

table with the first column representing the time and the other columns representing the values of the selected nodes. A XY-plot plots the data such that the values of one variable will be in the X axis and the other selected node values will be on the Y-axis. The X-axis represents by default the time, However it can be adjusted to represent a place or a transition of the Petri net. The final type is the histogram viewer and it shows the simulation results as a histogram where the frequencies of node values are plotted.

### 6.3.2 Associate a Viewer to a View

The default viewer of a view is the XY-plot. However you can change it by selecting the one you prefer from the viewers list box. Your changes will locally be saved when you navigate to another view.

### 6.3.3 Edit Viewer's Properties

When a viewer is selected, it loads its properties from the current view. These properties can be customised individually for each view. The number and type of these properties depend on the viewer type (i.e., tabular, XY-plot, or Histogram). To edit the properties of a viewer, do the following steps:

- Next to the viewer list box click edit.
- A multi-tab dialogue will open as illustrated in Figure 24.
- The number of tabs and properties will be different from one viewer to another.
- Adjust the properties and click Ok.

In the following, we list some of these properties and their meanings.

**Title** used to give a title to the view. This title is rendered on the viewer window. A view title can be set differently from a view name. New views get the view name as title. The title field applies to all the viewer types.

**Legend Tab** Applies for XY-plot and Histogram plot.

- **Show legend** Switch on and off the legend.
- **Horizontal position** Control the vertical position of a legend. Possible values are: right, centre, and left.
- **Vertical position**

  Control the horizontal position of a legend. Possible values are top, centre, and bottom.

- **Sort curves**

  Sort alphabetically the curve names in the legend position. Possible values are: ascending (sort the curve names from A to Z), descending (sort the curve name from Z to A), and not sorted (use the order in which the user checks the curves).

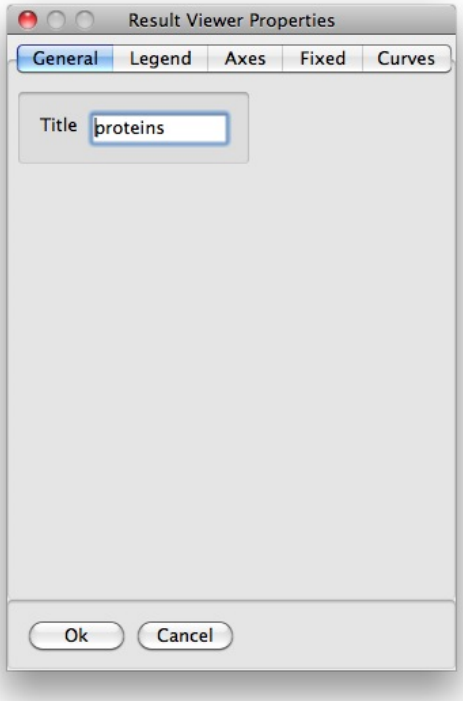



Figure 24: Adjusting viewer properties

**Axes Tab** Applies to the XY-plot and Histogram plot.

- **Show X-axis**

  Switch on and off showing the X-axis label.

- **X-axis label** Set the label of the X-axis

- **Show Y-axis** Switch on and off showing the Y-axis label.

- **Y-axis label** Set the label of the Y-axis.

**Fixed Tab** Applies to the XY-plot and Histogram plot.

- **Fixed X-axis adjustment** Switch on and off using a fixed interval for the X-axis.

- **Fixed Y-axis adjustment**

  Switch on and off using a fixed interval for the Y-axis.

- **Min. X value** Set the minimum value of the X-axis; the default is zero.
- **Max. X value** Set the maximum value of the X-axis, the default is one.
- **Min. Y value** Set the minimum value of the Y-axis; the default is zero.
- **Max. Y value** Set the maximum value of the Y-axis; the default is one.

**Curves Tab** Applies to the XY-plot.

- **Show lines** Switch on/off drawing the curves as lines.
- **Show symbols** Switch on/off drawing the curves as symbols.
- **Line width** Set the width of the line curve.
- **Line style** Set the style of the line curve.

**Bar Tab** Applies to the Histogram plot.

- **Bar width** Set the width of each bar.
- **Interval width** set the discretisation interval.
- **Frequency** Select the type of the histogram. Possible values are count or percentage.

## 6.4 Simulators

The different properties of the remotely running simulation can also be adjusted from the steering GUI. In this section we discuss some common tasks that can be done using the steering GUI.

### 6.4.1 Select a Simulator

After loading a model in the steering GUI, you might need to select a simulator to execute it. The currently running simulator (if there is any) is displayed at the runtime statistics panel. In the steering mode, a simulator has a type and a specific algorithm. Available simulation types are: continuous, stochastic, and hybrid. They represent the broad simulator categories. The available algorithms will depend on the selected simulator type. For instance, the hybrid simulation type has two different algorithms: static and dynamic.

To select a simulator to execute a model do the following steps:

- Select a simulator type from the simulator type drop box.
- Select an algorithm from the list of available algorithms.

Please note, the simulation will not start automatically after selecting a simulator. You will need to click the start button (see below).

### 6.4.2 Change Simulator Intervals

Before starting the simulation, you will need to enter the simulation interval. The simulation interval tells the simulator from where to start and stop the simulation. There are three items which you might need to change about the simulation interval:

- **Output start point** This is the point where the simulator should start to record the simulation output. The simulation will always start the simulation from time zero. However, the user might not be interested in the simulation output before a certain time point. Therefore you can tell the simulation about your preference. The default value for this item is zero.
- **Output end point** This is where the simulator will stop recording the output, and this also where it will stop the simulation. The default value is 100.
- **Output step size** It tells the simulator how often it should record the results.

### 6.4.3 Start the Simulation

Now you can start the remote simulation. In the lower panel you will find the "start" button. Hit start to begin simulating your model.

### 6.4.4 Refresh the Simulation Result

The simulation results as well as the runtime statistics are automatically refreshed during the simulation (see below). However, you can trigger a manual refresh when ever you want. To manually refresh the simulation results and the runtime statistics, hit the refresh button.

### 6.4.5 Other Useful Buttons

There are some other buttons which you can use during the simulation. They are:

- **Stop** stops the simulation.

- **Restart** stops the simulation and starts it again.
- **Pause** pauses the simulation.
- **Resume** resumes the simulation after the user paused it.
- **Setting** opens the local setting dialogue.
- **Close** closes the steering GUI, however, the remote simulation will remain running.

### 6.4.6 Adjusting the Simulation Speed

During the steering mode, you might need to slow down the simulation so that you have the chance to make changes to the running simulator. For this purpose you will find a slider bar in front of the simulation algorithm where you will be able to control the running simulation algorithm. Many different speed levels are available, however you will be able to define the speed interval between two successive speed levels (see Changing simulator setting section). To change the simulator speed, scroll the slider bar to the left or the right. The speed of one is the lowest speed level, while the speed of ten is the highest speed level. Please note that the value that you will change here will affect other users who connect to this model, since only one simulator is created for each model.

### 6.4.7 Change the Simulator Setting

There are a number of properties that can remotely be adjusted for a simulator using the steering GUI. Generally speaking, the number and type of these properties will depend on the selected simulation algorithm. Please note that your changes of the simulation properties will not take effect until the simulator is restarted. Moreover, these properties are global settings, i.e., changes related to these settings will affect all other users who are connected to this model. To change the simulation settings, do the following steps:

- Hit the setting button in front of the simulator algorithm.
- A new dialogue will appear as shown in Figure 25.
- Change the specific item(s) you want and press ok.
- The simulator setting will be sent to the server and they will take effect next time the simulator will start.

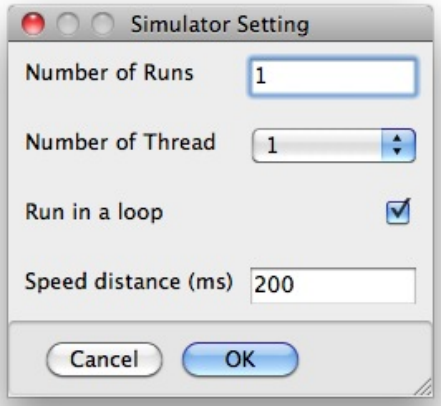



Figure 25: Simulator setting dialogue

## 6.5 Steering

In this section we discuss one of the distinguished features of the Snoopy steering framework: the ability to steer the simulation while it is running.

### 6.5.1 What Could be Steered

Through the Snoopy steering framework you can either change a value of a key parameter or a value of a certain place marking. The simulation will not stop during such a change, instead, it will continue execution from the point of time where the changes have been applied and any necessary re-initialisation will be done.

### 6.5.2 Selecting an Item to Steer

Before you perform the steering, you need to select a subset of the parameters or the places to steer their values during the simulation. However, you can also do this step during the execution, i.e., you will not need to stop the simulation or to do such a step in advance. To select a parameter or a place to steer its value, follow this procedure:

- From the edit parameter (place) sub window hit the button "Select".
- A new dialogue will open as shown in Figure 26.
- Select the parameters (places) of which you want to steer their values via the buttons in the middle of the dialogue.

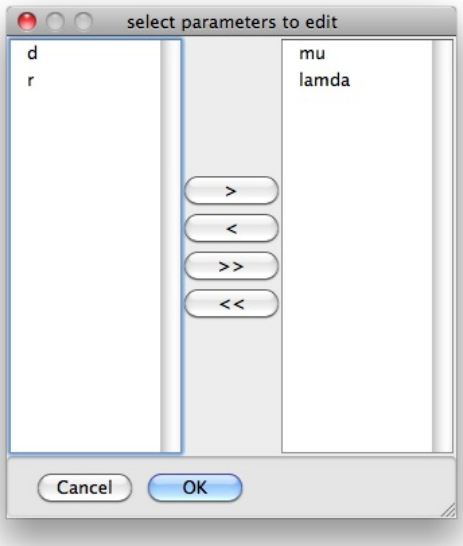



Figure 26: A dialog to select a parameter to steer

- Press Ok button.
- The selected parameters (places) will appear on the edit parameter (place) window as illustrated in Figure 27.

### 6.5.3 Change a Parameter (Place) Value

There are two methods to change a value of a parameter or a place during the simulation: you either increase or decrease the current value by a constant or you can enter a new value directly. In the former method, you can easily steer the simulation by just one mouse click and your changes will directly be propagated to the running simulation, while in the latter one you can enter a value with the precision that you want.

To increase (decrease) the current parameter (place) value by a constant follow these steps:

- Go to the parameter (place) of which you want to change its value.
- In front of the text box you will find a small spin button.
- Click on the up (down) button to increase (decrease) the value by a constant value.

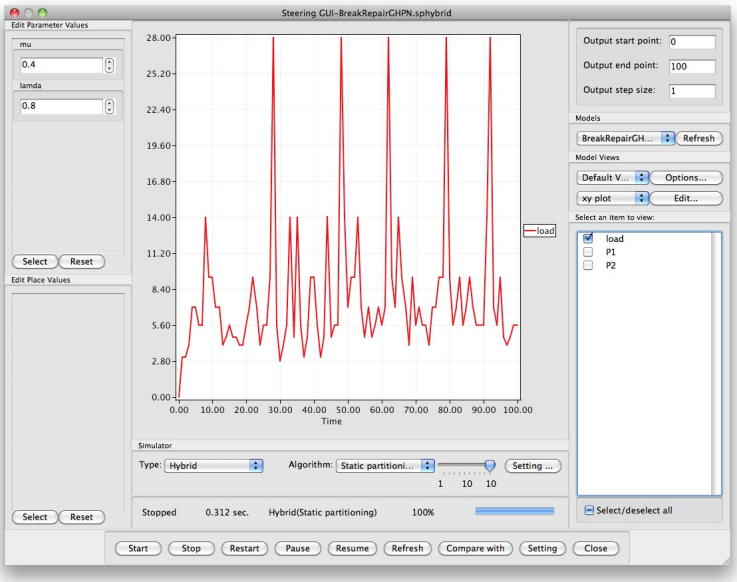



Figure 27: The steering GUI dialogue after selecting some parameters to steer their values

- The new value will directly be sent to the steering server without any further actions.

Please note that the default value for the increase (decrease) constant is 0.1, however you can change this value from the local setting button.

To completely enter a new value of a parameter or a place do the following steps:

- Go to the parameter (place) of which you want to change its value.
- Enter a new value and hit Enter.

### 6.5.4 Reset a Parameter (Place) Value

Sometimes it is useful to reset the simulation parameter values to the initial ones. Moreover, when you restart the simulation, it will restart using the current values. Therefore we added the option to reset the simulation parameter values or the current marking to the initial values. To do such a step, hit the reset button from the steering sub-window.

## 6.6 Local Settings

Using the Steering GUI you are able to change some local settings and your changes will only affect your client during the simulation and the steering. To open the local setting

window, click the "setting" button from the control button panel. A dialogue will appear with some options on it as illustrated in Figure 28. In the following we pinpoint these options.

### 6.6.1 Runtime Statistics Options

This set of options control refreshing runtime statistics which include simulation state, the simulation speed, simulation progress, etc. The "refresh periodically" option completely enables or disables refreshing the runtime statistics. Please note, if you disable this option, you will not receive any update from the server concerning the current simulation progress and the other statistics.

The second option is "refresh" where you can select a period of time after which these runtime statistics should be refreshed. The default value is to make a refresh each one second.

### 6.6.2 Results Options

Similar to the runtime statistics options, there are a set of options that control refreshing the results. The "refresh periodically" check box enables or disables refreshing the simulation results. Please note that if you disable this option, you will not receive an automatic update of the intermediate results.

You can also control the refresh period of the simulation results by selecting an appropriate value for it. Please note, this value should be big enough that it does not block the GUI. This situation could happen when there is a larger model where the steering GUI spends most of the time doing a refresh. Please see the next subsections to learn how you can let the steering GUI automatically control this refreshing time.

### 6.6.3 Timeout

The communication between Snoopy and the steering GUI is a synchronous one. That is when a user issues a command, the GUI will block until the server responses or a certain time period is elapsed. The timeout option allows you to control this period of time that the GUI should wait until it unblocks if the server does not response.

### 6.6.4 General Setting

Under this category you find some general options to control the execution of the steering GUI. One option for instance permits you to let the steering GUI determine the time

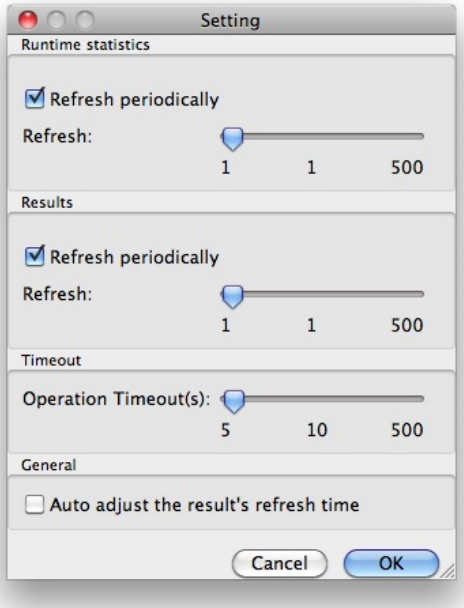



Figure 28: Local setting window

needed to refresh the intermediate results.

# 7 Steering Server

At the core of the architecture is the steering server. To support true collaboration between different users, the steering server is designed to be multi-user, multi-threaded, multi-model and multi-simulator. The server internally records information about users, model specification, as well as Petri net definition. Moreover, the server is shipped with a default set of simulators to permit the simulation of complex biological pathways without any additional components.

The mlti-user feature allows for more than one user to simultaneously share the same model and collaboratively steer the running simulation to get deeper insights of the problem under study. Furthermore, the multi-model and multi-threaded features coupled by multi-simulator capabilities of Snoopy render the concurrent execution of multiple models and flexible switching between different intermediate results.

The steering process takes place through an internal scheduler of the steering server. Each model has its own scheduler that coordinates the operation of the associated simulator. When a user submits a remote command from the GUI client, the current model

scheduler adds this command to what is called TO-DO list. Later on, when the simulator is ready to dispatch this command, the command is executed and its effect is displayed to peer users (i.e., collaborating users). The steering commands can be: altering model parameters, altering place marking, restarting, pausing, stopping the simulator, etc. The reason behind such organisation is that we can not steer the biochemical simulation at any point of execution, we have to wait for a suitable time point before the change can take place. Furthermore, using such an approach, the simulator does not need to wait for the user input and accordingly, any delay is eliminated due to the incorporation of computational steering into the simulation algorithm. The appropriate time point of a change depends on the simulation algorithm (i.e., continuous, stochastic, or hybrid algorithm). For instance, appropriate time points to change in continuous simulation are between integration steps of the ordinary differential equations. In case of conflicts between different users sending the same steering command to the same running model at about the same simulation time point (e.g., two users want to change model parameters at time 20), only the latest command will take effect and afterwards other users are informed of the decision.

In this section we discuss some of the options that are available at the server side.

## 7.1 Starting and Stopping the Server

In some situations you will need to stop the server and start it again.

### 7.1.1 Stopping the Server

To stop the server, i.e., to release the currently allocated IP and service, select from the server menu shutdown. The server state will be changed from running to stopped. Please note, this will not remove the running models or stop the simulation. It only closes the listening socket and it will not accept any further connections from the steering GUI clients. Besides, other connected clients will be disconnected, too. After this step, you can use the IP and the service number by other servers.

### 7.1.2 Starting the Server

To start it again, select from the server menu "start".

### 7.1.3 Restart the Server

To make a stop, immediately followed by a start, select "restart' from the server menu.

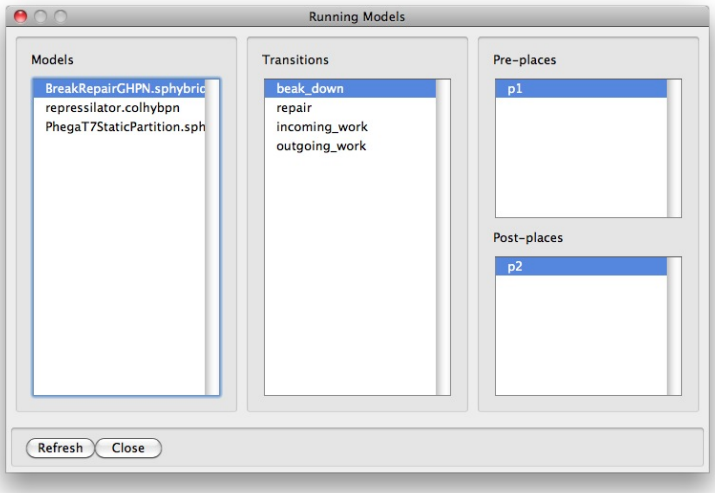



Figure 29: Model explorer window

## 7.2 Exploring the Running Models

At certain situations you might need to view the models currently running inside the server. To explore the running models, follow these steps:

- From the view menu select "Models explorer".
- A new dialogue will open as it is illustrated in Figure 29.

The new window has three columns. The first (from left to right) column lists the currently running models, the second column lists the transitions associated with that model, while the third one lists the pre-places and post-places that are associated with the selected transition.

Hovering the mouse over any of these items will provide you with additional information about it. For instance, hovering the mouse over the currently selected model will display the number of transitions and the number of places of this model using a tool tip control.

## 7.3 View Server Information

There are some information of the running server which can be viewed. In this section we list some of them.

### 7.3.1 Service Number

To view the listening service number (port) of the server, select from the view menu "listening port". A message box will appear with this information.

### 7.3.2 Listening IP

To view the current IP address of the server's machine, select from the view menu "view IP". The IP address will be displayed in a message box.

### 7.3.3 System Information

For diagnosis purposes you can view the system information, i.e., the current representation of the different data types. To do this, select "system information" from the view menu.

## 7.4 Saving and Loading Running Models

### 7.4.1 Save the Models

You can also save the currently running models to an XML file so that you can load them later. To save the running models, select from the file menu save. A dialogue box will appear where you can specify the file name and the path. The file extension is .sss. Please note that you will be asked to stop the server before saving the models into a file.

### 7.4.2 Load the Models

To load models from a file, select "load" from the file menu. A dialog box will appear where you can select a file to load the models from. Please note that saving and loading coloured models might take a long time as the unfolded version of the model is saved to the file.

## 7.5 Log Window

The log window is used to display information about the current activity of the server. In this section we list some of the operations that can be performed on the log window.

### 7.5.1 Clear the Log

You can clear the history of the log window by selecting "clear" from the "log" menu.

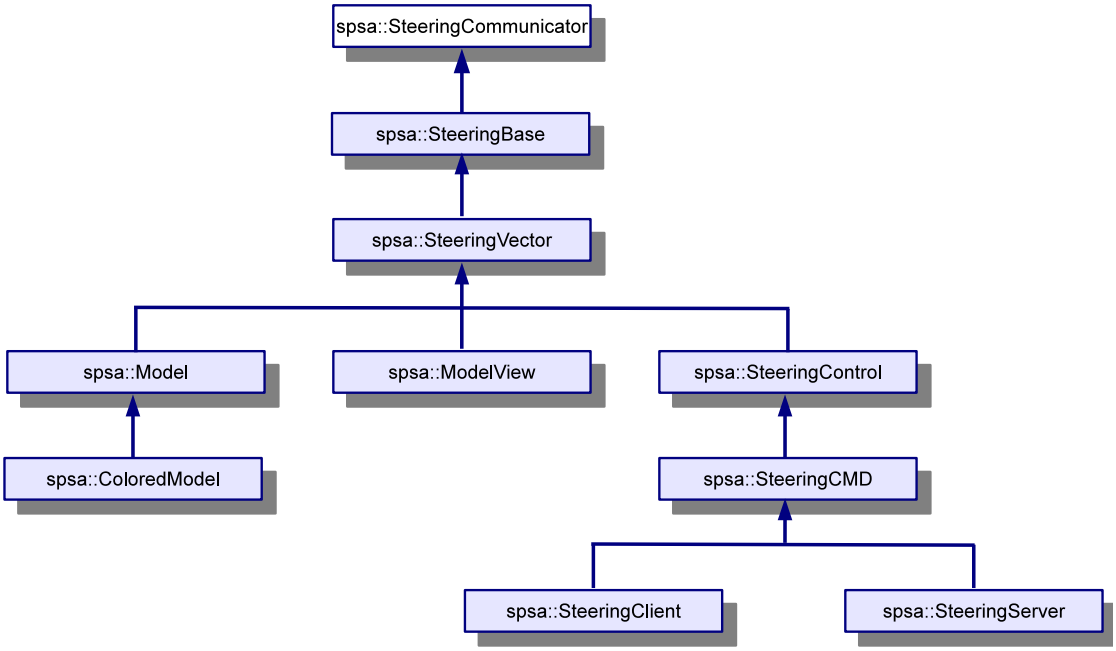



Figure 30: Inheritance diagram of Snoopy's steering APIs (SPSA). The Snoopy steering API classes can be classified into four main categories: communication, data structures, control commands, and end point components.

### 7.5.2 Save the Log

To save the information in the log window, select "save" from the "log" menu

# 8 Steering API

To keep the computational steering framework simple, yet extendible, an API library is of paramount importance. Modern software permits users to extend existing capabilities by adding new features or improving existing ones. Such extensions could be deployed using, e.g., plug-in or API calls. For our purpose, we adapt the concept of APIs to provide involved functionalities for advanced users. The main roles of the API library in our framework are: extension of the introduced framework to include additional simulators, communication between different framework components, and user ability to design a new user interface as well as visualisation modules that are compatible to communicate with other components. Figure 30 illustrates the different classes of our implementation of the steering API library.

While our framework comes with a set of full-fledged simulators, it is possible for

users to have their own simulation code included in the Snoopy framework. Snoopy's steering API library renders it possible to convert such batch simulation code into an interactive one.

Furthermore, the API library makes the entire design of the framework easy to be implemented and simultaneously promotes the reuse of existing code. For instance, the steering server and the steering GUI use the same API library to communicate with each other. Additionally, users are not confined to use the same user interface which is illustrated in Figure 14. Instead, they could implement their own dialogues and use their favourite visualisation libraries. The availability of such an API library ensures that the newly designed GUI is compatible to communicate with other framework components.

In terms of functionality, Snoopy's steering API can be grouped into four components: communication, data structures, control commands, and end point components. The communication part provides an easy-to-use and portable tool to send and receive data between two connected entities. The provided APIs send and receive simple data types (e.g., integer, double, or character), or compound types (e.g., 1D vector or 2D vectors). Moreover, the API library provides two special data structures to help organising the simulation code inside clients and servers: models and model views. The former ones are used to encapsulate the Petri net models including all the necessary information about places, transitions, and arcs, rate functions, initial marking?, while the latter data structure facilitates the organisation of result views inside individual models. Models can contain multiple views to provide more elaborated ways of understanding the simulation results. Please notice that models can be extended to include coloured information as illustrated in Figure 30.

Moreover, the API library contains a number of control commands. The control commands enable the user to start, restart, stop, and pause the simulation. They provide a way to manage the remotely running simulation. Additionally, changing parameter values or asking to refresh the current simulation output is also considered as a steering command.

In the following subsections, we discuss how the API library can be used to interact with the server. The routines discussed here can be used to write a GUI client that is compatible with the SSServer or to interact with the SSServer via other clients.

Please note, it is not viable to provide a full documentation for all classes and routines of the steering API library in this manual. Therefore, we just outline some important functionalities here. The full documentation will be provided in a separate file.

## 8.1 Notations

We adapt the following few coding standards for the use of the steering API library:

- **Namespace:** the *spsa* namespace is used to hide the library routines and data types from your program. For instance, to refer to some constant (e.g., *SUCCESS*) you might need to use it as *spsa::SUCCESS* unless you include this namespace in your file (which is not preferred). Hereafter, we sometimes discard the namespace but this should be clear for you.

- We redefine data types to facilitate making changes in one place. For example *spsa::Int* is the standard *int* data type.

- Type *long* has a known portability problem, therefore we define *spsa::Long* to be *long long* and *spsa::Ulong* to be *unsigned long long*. That is we use a 64-bit long data type on all platforms.

- We add a prefix to the used name to increase readability. This prefix denotes the role of the name and its data type. For example p_sHostName, means that this name will be used as a parameter (hence 'p') and it is of string data type (hence 's').

## 8.2 Initialising the Library

Before starting the actual communication between the server and the client, you need to initialise the steering library. This section briefly discusses how the steering library can be initialised.

### 8.2.1 Creating a Client

The *SteeringClient* class and its methods are mainly used to perform the communication between the SSServer and the user's client. Therefore, you will need to create an object of this class or to override its member functions in order to construct a client.

### 8.2.2 Connect to the Server

The first step to interact with the SSServer is to establish a connection. The following routine can be used for this purpose.

```
Int ConnectToServer(const String& p_sHostName, const String& p_sService, const Bool& p_bConnectionType)
```

- **Description:** tries to connect to a given computer name where the SSServer is running.
- Input:
  - p_sHostName: the computer full name or the IP address where the SSServer is running. Default is localhost.
  - p_sService: the service number to which the SSServer is listening. Default is 3000.
  - p_bConnectionType: specifies whether the routines should block until a connection takes place or not. Default is false.
- Return: Operation state: this can be either one of the following flags:
  - SUCCESS: the connection is established successfully.
  - FAILURE: there is a problem doing the connection.
  - INVALIDVERSION: the server version is different from the library version.

### 8.2.3 Initialise

The next step is to initialise the library before you can call any of the routines (except those which do not require initialisation, see Section 8.3.1). You have two options to initialise the Steering API library: either using a new model or using an existing one from the server (see Section 2.2).

```
Int Initialize(Model* l_pcModel)
```

Input: l_pcModel, a pointer to the model data structure (see the next section)

```
Int Initialize(const String& p_sExistingModelName)
```

Input: p_sExistingModelName the existing model name that you want to use.
Return:

- SUCCESS: the initialisation is done successfully.
- FAILURE: there is a problem doing the initialisation.

Please note, if you initialise the library using an existing model, the library will receive a copy from the server.

## 8.3 Dealing with Models

The steering API library provides two data structures to facilitate the construction and manipulation of models. They are *spsa::Model* and *spsa::ColoredModel*. A model is created by creating an object of either of these two classes and fill in the object with the Petri net definition and the metadata. Please note, after the initialisation, there will be a copy of the model inside the library, which is used during the initialisation.

### 8.3.1 Receive Model Names

To get the currently running model names, you can call the following function.

```
Int GetRuningModelNames(VectorString& p_asModelNames)
```

Return:

- p_asModelNames: a vector of strings containing the model names.
- SUCCESS: the initialisation is done successfully.
- FAILURE: there is a problem doing the initialisation.

Please note that you can call this function to get a list of model names before you call *Initialize* so that you can use one of these names as a parameter to *Initialize*.

## 8.4 Sending Commands

Similar to using the steering GUI to send commands to the server, you can use the steering API to call functions which send commands to the SSServer. The following is a list of these functions with a brief description.

- *CMD_Start*: starts the simulation.
- *CMD_Stop*: stops the simulation.
- *CMD_Restart*: restarts the simulation.
- *CMD_Pause*: pauses the simulation.
- *CMD_Resume*: resumes the simulation.

## 8.5 Monitoring and Steering

The monitoring is performed by asking the server to send the result in a matrix form, while the steering is performed by asking the server to change some parameter or values. In the following, we discuss both of them individually.

### 8.5.1 Monitoring

To read the current simulation results from the SSServer, you can call one function which receives the result in the form of a two dimensional (2D) matrix.

```
Int ReadResultMatrix(Vector2DDouble& p_anResultMatrix);
```

Return:

- p_anResultMatrix: a 2D matrix where the rows represent the recorded time points and columns represent the places or transitions.
- SUCCESS: the operation is done successfully.
- FAILURE: there is a problem receiving the result matrix.

### 8.5.2 Steering

**Change a Parameter value** To change a parameter value at the server side you can call the following function.

```
Int ChangeCurrentParameterValues(const Ulong& p_nParamPosition,const Double& p_nParamValue);
```

inputs:

- p_nParamPosition: The position of the parameter of which you want to change its value.
- p_nParamValue: the new value of the parameter.

Return:

- SUCCESS: the change is done successfully.
- FAILURE: there is a problem doing the change.

**Change a Marking value** To change a making value at the server side you can call the following function.

```
Int ChangeCurrentPlaceValues(const Ulong& p_nPlacePosition,const Double& p_nPlaceValue);
```

inputs:

- p_nPlacePosition: The position of the place that you want to change.
- p_nPlaceValue: the new value of the place marking.

Return:

- SUCCESS: the change is done successfully.
- FAILURE: there is a problem doing the change.

**Reset Parameter** You can also reset the values of the parameters or the marking to their initial values after making a change.

```
Int ResetMarkingValues();
```

resets the current making to the initial ones.

```
Int ResetParameterValues();
```

resets the parameter values to the initial one.
Return:

- SUCCESS: the reset is done successfully.
- FAILURE: there is a problem doing the reset.

### 8.5.3 Other APIs

Besides, those few functions which have been discussed in this section, the steering API library contains a rich set of APIs that deserve a special document.

# 9 Further Reading


[Her13] Mostafa Herajy. *Computational Steering of Multi-Scale Biochemical Reaction Networks*. PhD thesis, Brandenburg University of Technology Cottbus - Computer Science Institute, 2013.

[HH12a] M. Herajy and M. Heiner. Hybrid representation and simulation of stiff biochemical networks. *Nonlinear Analysis: Hybrid Systems*, 6(4):942–959, 2012.

[HH12b] M. Herajy and M. Heiner. Towards a computational steering and Petri nets framework for the modelling of biochemical reaction networks. In *proceedings of 21th international Workshop on Concurrency, Specification and Programming (CS&P 2012)*, volume 21. Humboldt University of Berlin, 2012.

[HHL+12] M. Heiner, M. Herajy, F. Liu, C. Rohr, and M. Schwarick. Snoopy – a unifying Petri net tool. In *Proceedings of 33rd International Conference on Application and Theory of Petri Nets and Concurrency*, volume 7347, pages 398–407. Springer, 2012.

# Appendices

## A Product Sheet

The product sheet summarises the most important features of the computational steering framework and the required system resources for its application.

It has been compiled for the user's convenience to allow for a quick assessment of the framework's potential usability.

Snoopy Computational Steering Framework, Product Sheet
http://www-dssz.informatik.tu-cottbus.de/

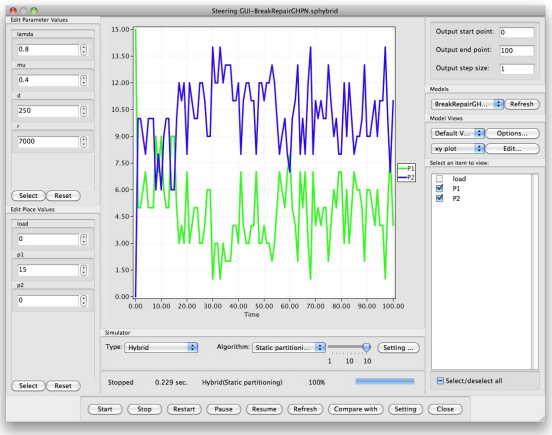
# Overview

The Snoopy steering framework consists of the Snoopy Steering Server (SSServer), the Steering Graphical User Interface (Steering GUI), the Steering Application Programming Interface (APIs), and the internal and external simulators.

In a typical application scenario, a user constructs a model using a Petri net editing tool (e.g., Snoopy). Afterwards, the Petri net model is submitted to one of the running servers to quantitatively simulate it. Later, other users can adapt their steering GUIs to connect to this model. One of the connected users initialises the simulation while others could stop, pause, or restart it. When the simulator initially starts, it uses the current model settings to run the simulation. Later, other users can remotely join the running simulation and change on the fly parameters and the current marking.

# Features

- Remotely run and control a simulation
- Execute the same model using different simulation algorithms
- Manage concurrently different models with possibly different simulators
- Define different views to explore the simulation results
- Explore on the fly your running models
- Steer simulation parameters while the simulation is running
- Control the simulation speed
- Connect to your simulation at any time from whatever place
- Collaborate with other people while executing model dynamics
- Platform-independent implementation



# Technical Specifications

- Client-server architecture
- Use of (colored) continuous, stochastic and hybrid Petri nets to construct and manipulate models
- Comes with full-fledged built-in simulators

# System Requirements

| Requirements | Snoopy | | SSServer | |
|---|---|---|---|---|
| | Minimal | Optimal | Minimal | Optimal |
| **Processor** | 1 GHz | 2 GHz | 2 GHz | 2.5 GHz or higher |
| **RAM** | 256 M | 1 GB | 512 M | $\geq$ 8 GB |
| **Free Hard Disk Space** | 500 M | 2 GB | 500 M | 10 GB |
| **LAN adapter** | X | X | X | X |

August 20, 2013

# B Quick Start

The quick start material includes everything one needs to know to give the computational steering framework a try. Actually, its contents is a subset of the complete user manual, which aims to give a comprehensive description of all features.

It has been compiled for the user's convenience to allow for a quick test of the framework without having to read the whole manual.

# Snoopy Computational Steering Framework
Quick Start
Version 1.0

Mostafa Herajy and Monika Heiner

– Data Structures and Software Dependability –
Institute of Computer Science
Brandenburg University of Technology
Cottbus, Germany

snoopy@informatik.tu-cottbus.de *

August 20, 2013

*Please sent all questions, comments and suggestions how to improve this material to this address.

## Contents

# 1 Introduction

With the advances of computing power and the proliferation of multi-core processors, it becomes essential to execute long running and computationally expensive simulations at powerful and remote computers – which enjoy high speed computational units – to profit from such precious processing resources. However, such powerful computers do not provide a direct interactive visualisation and analysis of the resulting simulation data due to either the intrinsic patch processing approach of these computers or the sharing of their resources between different users. Thus, there is a need to remotely mange and analyse the simulation output traces simultaneously while the simulation is in progress. Correspondingly, many different techniques have been proposed to overcome these limitations. Computational steering is among the elegant and promising tools that provide a tight coupling between simulation and visualisation modules of scientific models.

In this Quick Start manual we briefly discuss the use of Snoopy's computational steering framework to simulate and interactively steer Petri nets, e.g., biochemical network models. There are a lot of important functionalities that are provided by this framework in order to facilitate the conduction of "wet-lab" experiments. If you are wondering about what these tools can add to your work flow, it might be worth reading the following paragraphs.

The following are some aspects of what you can do using the Snoopy steering framework:

- **Remotely run and control a simulation**

  You can run your simulation at a remote computer. This feature allows the adaptation of high computational power machines from a local computer. The simulation will run at a server machine while the visualization of the results is done at a different machine, serving as a GUI client.

- **Execute the same model using different simulation algorithms**

  Sometimes it is useful to study a model in different paradigms, i.e., stochastic, continuous, or hybrid. In this framework, the same model definition can be executed with different simulation algorithms.

- **Manage concurrently different models with possibly different simulators**

  Different models can be simultaneously executed at the server side. Each model is assigned a separate simulator and can be executed independently from other



running models.

- **Define different views to explore the simulation results**

  Views provide a quick means to explore model results from different perspectives. Each view is defined by a set of curves and their associated attributes. Different views can be defined for the same model.

- **Explore on the fly your running models**

  Using the steering graphical user interface (Steering GUI), you can easily navigate among different models. Besides, the list of running models at the server side can be refreshed if another user adds a new model to the server.

- **Steer the simulation parameters while the simulation is running**

  The main goal of Snoopy steering framework is to enable users to interact with their models during simulations. Users can change model parameters as well as current marking and immediately monitor the system's response for such changes. This is an useful tool since a user is allowed to ask "what-if" questions.

- **Control the simulation speed**

  The simulation speed can be set to an appropriate level to facilitate the interaction with a running model during its execution. This feature is important if the simulation parameters are allowed to be changed while the simulation is running.

- **Connect to your simulation at any time from whatever place**

  The overall organization of this framework is flexible to let users connect/disconnect to/from running models without affecting their execution. Moreover, you can connect to your models from different places, for example from your office or from your home.

- **Collaborate with other people while executing model dynamics**

  More than one user are permitted to connect to the same models. Users can collaborate in executing and steering a running simulation. This feature might promote the sharing of knowledge between different users with different backgrounds.

- **Platform-independent implementation**

  The core communication library is written in standard C++ and therefore it can run on different platforms among them are windows, Mac OS X, and Linux. Moreover, the client and server does not need to run under the same platform.

## 2 Overview

In this section we give a general overview of the Snoopy steering framework to help understanding the high level organization before going into details.

### 2.1 Framework

Figure 1 presents the general architecture of the Snoopy steering framework. Its main components are: the steering server, the steering graphical user interface, the steering application programming interface (APIs), and the internal and external simulators. These interdependent ingredients enable users not only to run their biochemical network models and obtain results, but also to share, distribute and interactively steer them. Additionally, users do not need to wait until the simulation ends in order to discover potentially incorrect results. Instead, using this framework, errors could be discovered early and be immediately corrected during the simulation and, if necessary, the simulation could be restarted using the current setting. Subsequently, the overall time required to carry out dry-lab experiments will substantially decrease.

The main component of the architecture is the steering server. It is the central manager of the model data and communication traffic between the different framework components. It is a multi-user, multi-model, multi-simulator, and multi-threaded server. Inside the server, data is organised in terms of individual models which are defined by means of Petri nets.

The steering graphical user interface is the user's entry point to the overall architecture. Through it, the user can monitor and steer the simulation output and the corresponding key parameters, respectively. Users can flexibly connect and disconnect from their local machines to the available steering servers and view the currently running models. Model dynamics are produced using either an internal or an external simulator. Internal simulators are implemented inside the server which currently supports deterministic, stochastic, and hybrid algorithms, while external simulators are defined by the user and dynamically linked to the running server.

The steering application programming interfaces (APIs) are used to incorporate external simulators into the steering server. Additional responsibility of the API library is to facilitate the connections between the different framework components. More specifically, it is used to carry out the communication between the steering GUI and the steering server.

Finally, this framework permits the simulation to be remotely executed using an external simulator developed by the user (optional component). The communication



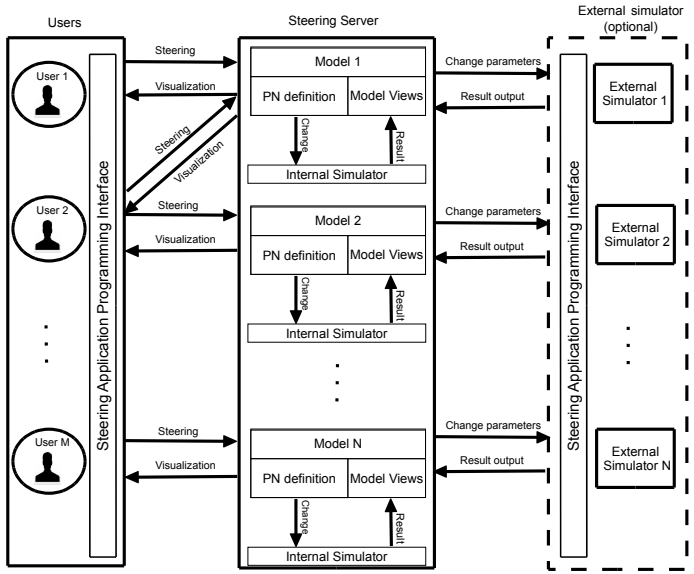



Figure 1: Petri nets and computational steering framework. The framework consists of four components: steering server, steering graphical user interface (GUI), steering application programming interface (Steering API), and simulators (internal and external). The flow of information goes in two opposite directions: from the simulator to the user (monitoring) and from the user to the simulator (steering). The data structure inside the server is organised in terms of Petri nets: places, transitions, arcs and parameters. The place data structure includes the initial marking, and the transition data structure the rate functions. A model can contain different result views, which are defined by the users and submitted to the server for further download and manipulation.

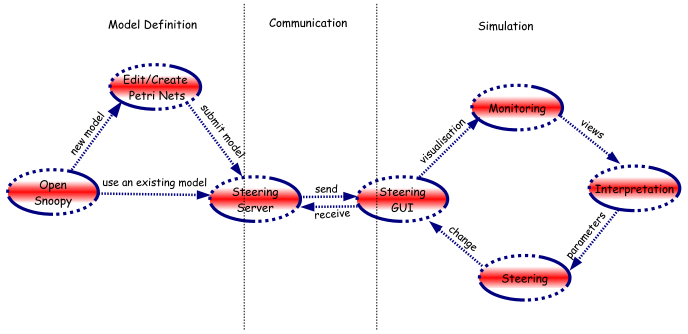



Figure 2: Graphical illustration of a typical application scenario of Snoopy's steering framework. The user has two options at the beginning: either reading one of the models already existing in the server or submitting a new model. In the latter case the Petri net model can be created using Snoopy or other Petri net editing tools by use of our API library. Afterwards, Snoopy's steering GUI can be used to perform the monitoring and steering.

between these external simulation modules and the other architecture components takes place through the steering APIs. This means that with modest effort, users can include their own favourite simulators and perform the monitoring and steering tasks by help of the other framework components.

### 2.2 Application Scenario

In a typical application scenario, a user constructs the biochemical reaction network using a Petri net editing tool (e.g., Snoopy). Afterwards, the (stochastic, continuous, hybrid) Petri net model is submitted to one of the running servers to quantitatively simulate it. Later, other users can connect to this model by their steering GUIs. One of the connected users initialises the simulation while other users could stop, pause, or restart it. When the simulator initially starts, it uses the current model settings to run the simulation. Later, other users can remotely join the simulation and change model parameters or the current marking. Figure 2 illustrates graphically a typical application scenario of Snoopy's computational steering framework.



Table 1: Minimal and Optimal Hardware Requirements.[0]

| Requirements | **Snoopy** | | **SSServer** | |
|---|---|---|---|---|
| | Minimal | Optimal | Minimal | Optimal |
| Processor | 1 GHz | 2 GHz | 2 GHz | 2.5 GHz or higher |
| RAM | 256 M | 1 GB | 512 M | $\geq$ 8 GB |
| Free Hard Disk Space | 500 M | 2 GB | 500 M | 10 GB |
| LAN adapter | X | X | X | X |

## 3 System Requirements

In this section we outline the hardware and software requirements to run Snoopy and the Snoopy steering Server (SSServer) on your computer.

### 3.1 Hardware Requirements

The hardware requirements depend on your specific needs. For instance, if you plan to run big models (100,000 to 1000,000 variables), then higher requirements are needed than to run relatively small ones. At the time of writing this manual, a 64Bit machine, preferably running native Linux is recommended for computational expensive models. Table 1 outlines the minimum and the optimal hardware required to run the Snoopy steering framework.

### 3.2 Software Requirements

The Snoopy steering framework implementation is platform-independent. Therefore you can use it on your favourite operating system. In more specific terms, you will need one of the following operating systems running on your computer:

- Window XP or higher
- Mac OS X 10.5 or higher
- Linux, e.g., Ubuntu

[0]This table is both for 32 and 64 Bit machines.

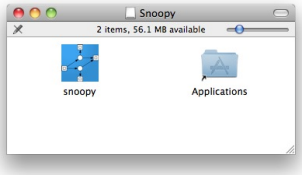



Figure 3: Snoopy Setup

Next, you will need a copy of Snoopy suitable for your operating system which can be downloaded from `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy`.

Finally, a copy of the SSServer is required to run this framework. Please note, at the time of writing this manual, the SSServer is not available at Snoopy's website. However, it can be requested from the authors by email.

# 4 Installation

The specific installation procedure depend on your operating system version. Below we give two examples using Windows and Mac OS X.

## 4.1 Install Snoopy under Mac OS X

To install Snoopy under Mac OS X, first acquire a Snoopy version. The Snoopy setup file under Mac OS is in a disk image format (dmg). All the necessary data are provided in a single file. To install this file on your computer do the following steps:

- Mount the disk image on your computer by double-clicking the dmg file.
- The disk image will appear as another CD in your Finder with the name "Snoopy".
- Opening this disk, you will find the Snoopy application bundle, as shown in Figure 3.
- Copy Snoopy to your desired location, or drag and drop it into the application folder.



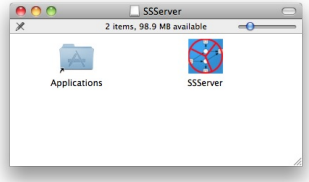



Figure 4: SSServer Setup

## 4.2 Install SSServer under Mac OS X

To install the SSServer to your Mac OSX, similar steps as for installing Snoopy are required. We repeat them again for your convenience.

- Mount the disk image on your computer by double-clicking the dmg file.
- The disk image will appear as another CD in your Finder with the name "SSServer".
- Opening this disk, you will find the SSServer application bundle, as shown in Figure 4.
- Copy the SSServer to your desired location, or drag and drop it into the application folder.

## 4.3 Install Snoopy under Windows

To install Snoopy under Windows, follow these steps:

- Obtain the windows installer package for Snoopy. This should be a msi file.
- Double click the obtained setup package.
- The windows installer will start as shown in Figure 5.
- Follow the simple instructions in this window by hitting the next button.
- At the end of these steps Snoopy should be installed on your computer and a shortcut will be created at your desktop.
- To start Snoopy double click the Snoopy's shortcut from your desktop.


Figure 5: Snoopy Setup under Windows

### 4.4 Install SSServer under Windows

Similar steps are required to install the SSServer under Windows. For your convenience we repeat them in the following procedure.

- Obtain the windows installer package for SSServer. This should be a msi file.
- Double click the obtained setup package.
- The windows installer will start as shown in Figure 6.
- Follow the simple instructions in this window by hitting the next button.
- At the end of these steps SSServer should be installed on your computer and a shortcut will be created at your desktop.
- To start the SSServer, double click the SSServer's shortcut from your desktop.

## 5 Getting Started

### 5.1 Launching Snoopy and SSServer

The first step to use Snoopy in the steering mode is to open the Snoopy Steering Server (SSServer) on your local computer or on another remote machine where you want to run the simulation. Second you need to open Snoopy on your machine. The exact procedure



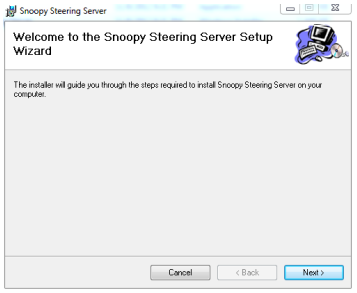



Figure 6: SSServer Setup under Windows

depends on the operating system you use. We assume that you have basic knowledge to run software on your operating system.

After starting the server, it will look as shown in Figure 7 under the Mac OS X. Other platforms might have slightly different, but similar interfaces. There are two important information you need to notice after starting the SSServer: the running server name (IP) and the listening service number. They are written in the log window after the SSServer started successfully. You will need these information when connecting to a server using Snoopy. For instance, in Figure 7, the server name is swqlab3.informatik.tu-cottbus.de and the IP is: 141.43.202.57. The server is listening to the service number 3000.

In the status bar, there are some useful information about the status of the SSServer. The first one gives information about the current state of the server. For example in Figure 7, the SSServer state is "Running". Other possible states are "stopped" and "Restarted". Make sure before connecting to a server that its state is "Running". The second information in the status bar is the number of currently running models inside the server. Each time you submit a new model, this number will increase. The third and final information is the number of clients (users) that are currently connected to the server.

After you have started the SSServer, you need to open Snoopy on your machine. Figure 8 gives a screenshot of Snoopy under Mac OS X.

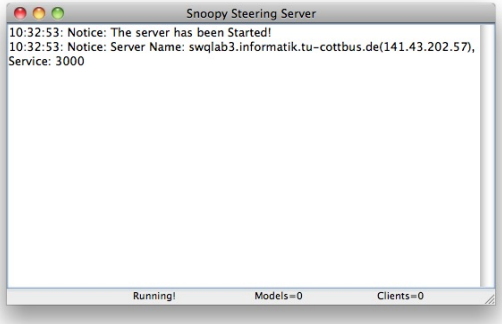



Figure 7: Snoopy Steering Server under Mac OS X

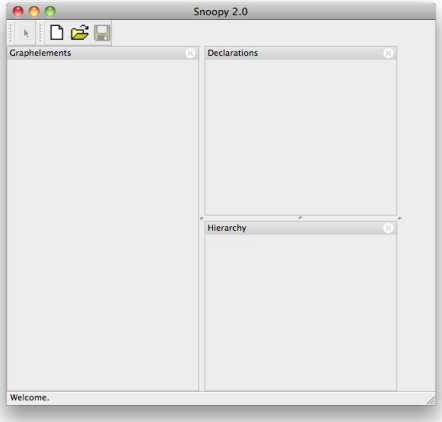



Figure 8: Snoopy under Mac OS X




Figure 9: Open Snoopy in the steering mode (here: with no Petri net open)

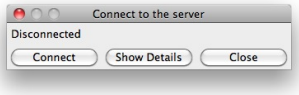



Figure 10: Connection window

### 5.2 Connecting to the Server

To connect to a running SSServer using Snoopy, you first need to decide if you would like to use a model from those running on the SSServer (if there are some) or if you would like to submit a new model. In the latter case you will need to open your Petri net model using Snoopy. Nevertheless, to open the connection window, do the following steps in either of these cases:

1. From the view window, select "Start Steering-Mode" (see Figure 9) or just press "F7" from the function keys if you prefer using short cut keys.

2. The connection window will be opened (see Figure 10).

After you have opened the connection window you can connect to the sever using the default setting via the "connect" button. This makes sense if you run the SSServer at the same computer as you use Snoopy or if you already configured Snoopy to connect to a certain SSServer. However, if you run the SSServer on another computer, you must first configure the connection before pressing the "connect" button, see Section 5.1.

Another important point is that the SSServer and Snoopy should have the same version number of the communication library. You cannot let Snoopy connect with an SSServer having a different version. A check will be done at the beginning and you will be informed if an incompatible version is detected. In this case the Snoopy and SSServer versions are displayed for your convenience.

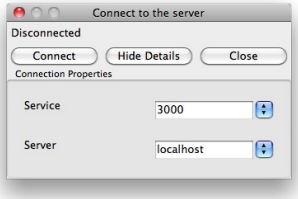



Figure 11: Configuring connection setting

### 5.2.1 Configuring Connection Settings

To configure the connection settings to connect to a certain SSServer do the following steps:

- Press the "Show details" button from the connection window.
- New fields appear as illustrated in Figure 11.
- In the service text box, enter the service number as discussed in Section 5.1.
- In the server text box, enter the server name or the IP address.

The default value of the service number is "3000", and the default for the server name is "localhost". To simplify entering these information, you can also select from previous setting using a list.

### 5.2.2 Saving Connection Settings

The connection settings, which you have entered in the previous step, are automatically saved to the Snoopy registry. Therefore you do not need to re-enter them again when you later want to re-connect to the same SSServer.

## 5.3 Submit a New Model

To use a new model in the steering mode do the following steps:

1. Open the Petri net file in Snoopy.



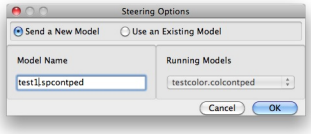



Figure 12: The steering options window

2. Open the connection window as discussed in Section 5.2.
3. Make sure that you correctly configured the connection settings to connect to a running server.
4. Hit the "Connect" button. The steering option window will appear as illustrated in Figure 12.
5. Make sure, the "send a new model" option is selected.
6. Enter a model name and then press ok.

Please note that the name of the loaded file in Snoopy is used as a default name for the new model. You can edit the name as you want under the condition that there is no model running on the SSServer side with the same name.

After pressing the Ok button, the steering dialogue will open as shown in Figure 13.

Please note, if you use a coloured model, then the unfolding window will automatically appear after step 6, because you need to unfold the model before submitting it to the server. To unfold a model, press the "Start" button on the unfolding window as shown in Figure 14.

## 5.4 Using an Existing Model

The Snoopy steering framework gives you the chance to use existing models that are running on the SSServer without opening them in Snoopy. For example, let us assume that you have one model that is already running in the SSServer and you want to disconnect and return back to it later. In this case, you do not need to reopen the Snoopy file or even to unfold it if it is a colored model. To steer an existing model, follow this procedure:

1. Open Snoopy, and then open the connection window as described in Section 5.2.

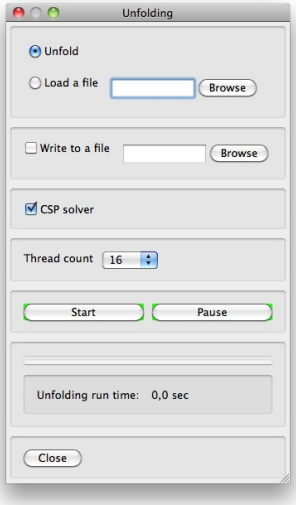



Figure 13: Automatic unfolding window

2. Press the "Connect" button.

3. A new window will appear with the models currently running on the SSServer.

4. Select one of these models and press ok.

5. The steering dialogue will open with the selected model loaded.

Please note, if there are no running models on the server side, then you will be informed and the process of opening the steering GUI will terminate. Besides, you can do these steps while a Petri net model is opened in Snoopy. However, you will need to select the option "Use an Existing Model". After that you can select one of the running models from the list.



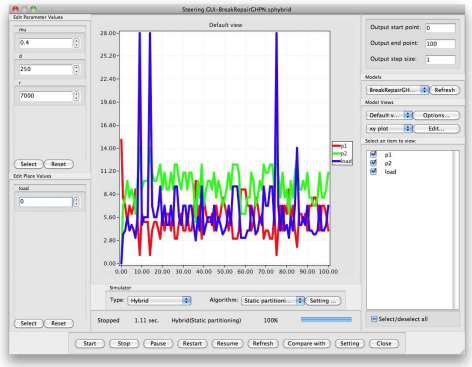


Figure 14: The steering GUI

### 5.5 Replacing an Existing Model

When constructing a Petri model for the first time, you might need to submit the same model several times with some modifications. Such modifications might involve adding new places or new transitions to the model definition. In this case, submitting the same model with a different name will increase the number of models running on the SSServer. A better procedure is to submit the modified model with the same name as the old one. In this case the modified model will replace the previous one. Please note, in order to replace an existing model, there should be no other users connected to it. If there are other users connected, the server will refuse to replace this model, and the client will get a notice.

## 6 Further Reading

[HH12a] M. Herajy and M. Heiner. Hybrid representation and simulation of stiff biochemical networks. *Nonlinear Analysis: Hybrid Systems*, 6(4):942–959, 2012.

[HH12b] M. Herajy and M. Heiner. Towards a computational steering and Petri nets framework for the modelling of biochemical reaction networks. In *proceedings of 21th international Workshop on Concurrency, Specification and Programming (CS&P 2012)*, volume 21. Humboldt University of Berlin, 2012.

[HH13] M. Herajy and M. Heiner. *Snoopy Computational Steering Framework - User manual (in preparation)*. Brandenburg University of Technology at Cottbus, Data Structures and Software Dependability, Cottbus, Germany, 2013.

[HHL+12] M. Heiner, M. Herajy, F. Liu, C. Rohr, and M. Schwarick. Snoopy – a unifying Petri net tool. In *proceedings of 33rd International Conference on Application and Theory of Petri Nets and Concurrency*, volume 7347, pages 398–407. Springer, 2012.