*Journal of Statistical Software*

*June 2012, Volume 49, Issue 3.* *http://www.jstatsoft.org/*

# Glotaran: A Java-Based Graphical User Interface for the R Package TIMP

**Joris J. Snellenburg**
VU University Amsterdam

**Sergey P. Laptenok**
VU University Amsterdam

**Ralf Seger**
Augsburg University

**Katharine M. Mullen**
National Institute of
Standards and Technology

**Ivo H. M. van Stokkum**
VU University Amsterdam

**Abstract**

In this work the software application called **Glotaran** is introduced as a Java-based graphical user interface to the R package **TIMP**, a problem solving environment for fitting superposition models to multi-dimensional data. **TIMP** uses a command-line user interface for the interaction with data, the specification of models and viewing of analysis results. Instead, **Glotaran** provides a graphical user interface which features interactive and dynamic data inspection, easier – assisted by the user interface – model specification and interactive viewing of results. The interactivity component is especially helpful when working with large, multi-dimensional datasets as often result from time-resolved spectroscopy measurements, allowing the user to easily pre-select and manipulate data before analysis and to quickly zoom in to regions of interest in the analysis results. **Glotaran** has been developed on top of the **NetBeans** rich client platform and communicates with R through the Java-to-R interface **Rserve**. The background and the functionality of the application are described here. In addition, the design, development and implementation process of **Glotaran** is documented in a generic way.

*Keywords*: **Glotaran**, **TIMP**, global analysis, target analysis, time-resolved spectroscopy, Java.

## 1. Introduction

**TIMP** was introduced as an R package for modeling multi-way spectroscopic measurements by Mullen and Van Stokkum (2007). It was born out of the need for an open-source, platform-independent and extensible problem-solving environment for fitting a wide range of models to

multi-way spectroscopy data, i.e., spectra measured as a function of one or more independent experimental variables such as time, pH, pD, temperature, excitation wavelength or quencher concentration. In addition it has been used in the analysis of microscopy data by Laptenok, Mullen, Borst, Van Stokkum, Apanasovich, and Visser (2007) and Laptenok, Borst, Mullen, Van Stokkum, Visser, and Van Amerongen (2010). **TIMP** has a very flexible command line interface, typically used in a scripted manner, that enables the manipulation of data, specification of the model(s), and viewing of the results through a few user-accessible functions. For the user the primary difficulty in operating **TIMP** lies in knowing the syntax of these functions and their many options. Another difficulty is that all steps of the analysis are specified in advance: The data to be read, what preprocessing is to be done (such as averaging, window selection or baseline-subtraction), what model to use and which traces and spectra to plot. There is little to no (visual) feedback until the analysis has successfully completed. The results produced then are relatively static; zooming is possible to an extent but with limited resolution, and if a region of interest falls outside of the selection of plotted traces then the plot command needs to be re-run with different parameters. In order to address these issues, a graphical user interface (GUI) for **TIMP**, called **Glotaran**, was created.

## 1.1. Introducing Glotaran

**Glotaran** was designed to extend **TIMP** with: (1) Interactive data exploration, allowing a user to interactively view all data rather than make a selection upfront, (2) easier (visual) modeling, to assist the user in building complex models without making typographical mistakes, and (3) interactive visualization of analysis results, to quickly assess the quality of fit. **Glotaran** stands for global and target analysis, the method by which these measurements can effectively be modeled and analyzed as described by Van Stokkum, Larsen, and Van Grondelle (2004). Development on **Glotaran** started at the physics department of the VU University Amsterdam as a GUI for the R package **TIMP** as reported by Snellenburg, Van Stokkum, and Mullen (2008). As such, **TIMP** still provides the required mathematical framework and computational algorithms for modeling and data analysis. In short, **Glotaran** was designed to exploit the functionality already existing in **TIMP**, address certain limitations (dynamic, interactive charts rather than static plots), provide new functionality (interactive data exploration and result viewing, and visual modeling) and simplify the overall process of model-based data analysis by providing an easy-to-use GUI. **Glotaran** aims to lower the barrier to access the advanced analysis and modeling capabilities of **TIMP** by eliminating the need to learn R syntax (R Development Core Team 2012) or specific **TIMP** code, while at the same time maintaining compatibility between models written in **TIMP** and models designed in **Glotaran**.

## 1.2. Background and motivation

Development of **Glotaran** was fueled by the desire to provide an easy-to-use modeling and data analysis tool for use by scientists in the field of molecular biophysics. These scientists use state-of-the-art spectroscopy and microscopy measurement techniques to study the mechanisms underlying the behavior of complex bio-molecular systems, often resulting in large and complex time-resolved spectroscopy and microscopy datasets. Because of the complexity of the physical processes underlying these data, advanced modeling and data analysis techniques such as global and target analysis are often necessary to analyze and derive models from these

measurements, as explaind by Van Stokkum *et al.* (2004). The release of **TIMP** in 2007 was the first step in the realization of an open-source, platform-independent and extensible data analysis and modeling tool for use with such data. However, **TIMP** – just like R – is limited to a command-line-interface, which means that using it requires knowledge of R syntax and **TIMP**-specific function calls and their arguments. Developing a full-fledged graphical user interface for an R package such as **TIMP** is no easy task, demonstrated by the wide variety of different R GUI solutions that exist (Grosjean 2010). As introduced in this special issue by Valero-Mora and Ledesma (2012), nowadays there are sophisticated front-ends for R such as **RKWard** (Rödiger, Friedrichsmeier, Kapat, and Michalke 2012) and **Deducer** (Fellows 2012) – an extension of **JGR** (Helbig and Theus 2005) – which feature integrated plugin support, facilitating easier creation of GUI's for new and existing R packages. For us the primary reason to take a different approach was the lack of a mature integrated development environment (IDE) for R such as is available for Java and the specific set of requirements that we set out for the project.

In this paper we address various design considerations in GUI development for an R package. In Section 2 we discuss the features of **Glotaran** along with an example case study and make the comparison to working with **TIMP**. In addition we address the issue of the visualization of data, model and analysis results. In Section 3 we define the specific application requirements we set out for the project and explain the design choices we made. In Section 4 we have documented the resulting applications' architecture, and discuss the complications of interfacing with R. Finally, the availability of the application and a discussion on future work can be found along with some concluding remarks in Section 5.

## 2. The application

**Glotaran** is a Java-based rich client platform (RCP) application that has been built on top of the **NetBeans** RCP (Böck 2011). As such the application consists of a collection of modules that each provide various aspects of the functionality of the application as a whole. In addition to the basic functionality provided by the platform's own modules, **Glotaran** comes with various modules of its own providing the higher level functionality that turns it into a useful application. Below is a list of the application's core functionality. Most of these features are illustrated in the screenshots that follow.

- Support for a well defined project structure consisting of a main project folder with sub folders containing datasets, models, analysis schemes, results and (optionally) simulation input files.
- Support for reading in various known data file formats, either in plain text or binary format.
- Interactive data exploration by means of a custom designed data editor, specific to the type of data shown.
- A visual modeling tool, or analysis scheme editor to specify a model for analysis, link it to the datasets to which the model should be applied and specify the starting parameters for analysis and configure the required run time parameters such as the number of iterations the analysis should run for.

- Functionality that connects to a running R process, translates the models to R function calls to **TIMP** and retrieves the results of the computations after analysis in **TIMP** has completed.
- Interactive results inspection by means of a custom designed editor, specific to the type of data shown.

The modularity of the application means that in theory much of the application's functionality can easily be extended, replaced or stripped out, simply by removing, adding or changing the existing modules, without breaking the application as a whole, as long as the individual modules follow good coding practices and depend only on publicly accessible application programming interfaces (APIs) from the **NetBeans** platform or **Glotaran**.

## 2.1. Interactive data exploration

**Glotaran** currently supports two types of data: Time-resolved spectroscopy data and time-resolved microscopy data. Both data-types differ in the way they should be presented to the user and therefore have two different visualization editors. Throughout this paper the spectroscopy data editor is shown as an example.

The spectroscopy data editor currently has three tabs: The first `Data` tab shows the actual data, the second `SVD` tab contains the tools to perform and analyze the singular value decomposition of the data-matrix and the third `Info` tab shows some general properties of the data-file when available. The data editor allows for some basic pre-processing of the data such as window selection, averaging and sampling, baseline subtraction and outlier detection. Because the data are directly visualized, and because the user can scroll through the data and zoom in on interesting regions, the pre-processing becomes much easier than in **TIMP** where one often needs to make an educated guess as to what region is of interest and, for instance, whether baseline subtraction is necessary.

Time-resolved spectroscopy data are measured as a function of the experimental spectral variable wavelength $\lambda$ and the independent experimental variable time $t$ relative to the instant of excitation. The model underlying the data matrix $\Psi$ is a superposition of $n_{comp}$ components given by the equation

$$\Psi(\lambda, t) = \sum_{l=1}^{n_{comp}} c_l(t)\epsilon_l(\lambda) \tag{1}$$

where $c_l$ and $\epsilon_l$ are the unknown concentration profile and spectrum of component $l$ respectively. Figure 1 shows the time-resolved spectroscopy data-editor with an example of such data.

In this editor the value of the measurement (fluorescence intensity, absorption, etc.) is shown as a function of time (on the vertical axis) and wavelength (on the horizontal axis). The sliders can be manipulated to show one specific time-trace (at a certain wavelength) or spectrum (at a specific time) from the entire dataset as indicated by the overlaying crosshair lines. This allows the user to scan through large datasets quickly and investigate particularly interesting regions easily. In Figure 1 we observe both positive and negative difference absorption (color changes) which decay over time. This indicates that an excited state is formed and decays in less than one nanosecond. The aim of this experiment is to discover the mechanism of this fast decay (see Hippius *et al.* 2007).

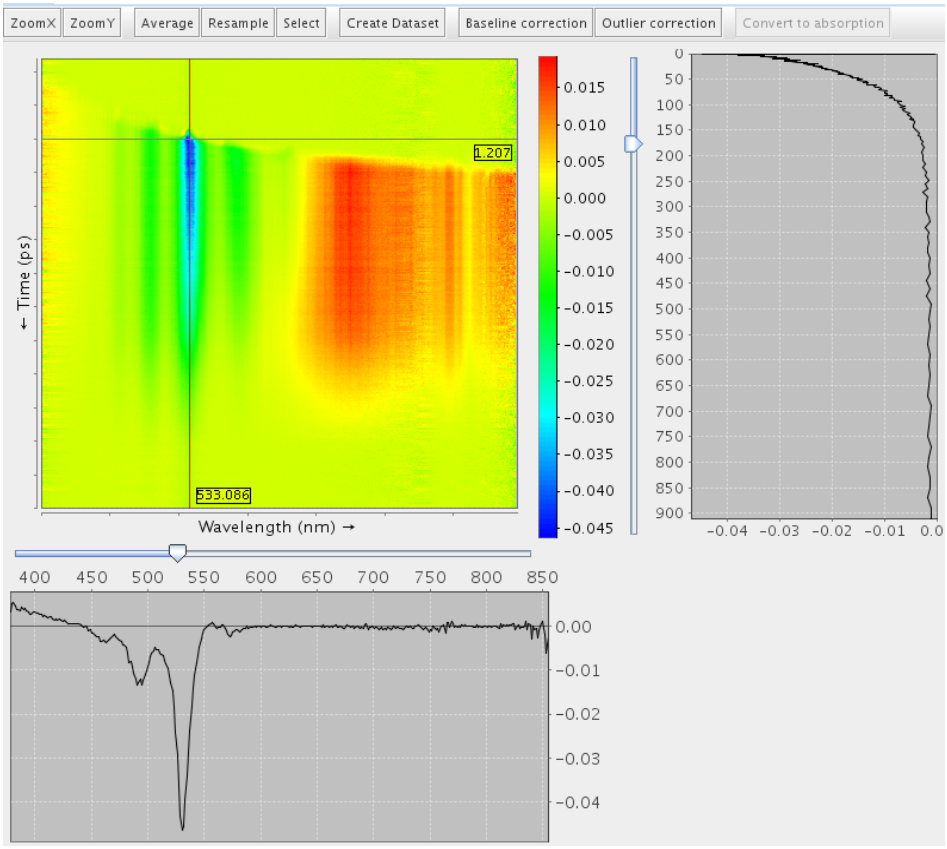


Figure 1: Time-resolved spectroscopy data editor; showing an example of femtosecond transient absorption spectroscopy data (from Hippius *et al.* 2007, Figure 5).

### *Singular value decomposition*

The first step in modeling these data typically involves looking at the singular value decomposition of the dataset. The singular value decomposition (SVD) is a matrix factorization technique which can be used to explore the number of spectrally and temporally independent components in the data matrix (Golub and Van Loan 1996), which is an important aspect of defining an initial model. The SVD of the data-matrix is defined as:

$$\Psi(\lambda, t) = \sum_{n=1}^{n_{max}} u_n(t) w_n(\lambda) SV_n \tag{2}$$

where $n_{max}$ is the minimum of the number of rows and columns.

The singular values give an indication of the number of independent components in the data. With $n_{comp}$ independent components and noise-free data there would be exactly $n_{comp}$ significant singular values (different from 0), defined as: $SV_1 \geq SV_2 \geq SV_{n_{comp}} > SV_{n_{comp}+1} = \ldots = 0$. In the case of data with a small amount of noise the significant singular values are no longer as clearly defined but typically still stand out from the rest. This is best captured in the form of a screeplot where the singular values are plotted on a logarithmic range axis in decreasing order, see Figure 2 (top panel). Furthermore, the accompanying $n_{comp}$ left and right singular vectors are clearly structured, see Figure 2 (left and right panel respectively).

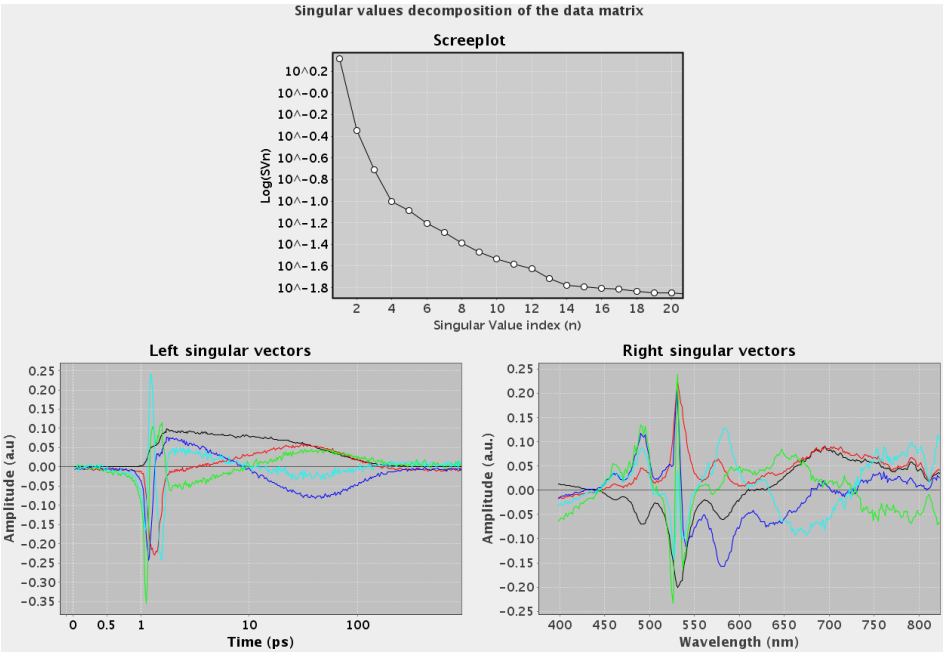


Figure 2: Singular value decomposition of the data-matrix of the dataset from Figure 1. Top: Screeplot. Bottom: The first five left and right singular vectors are plotted respectively in black, red, blue, green and cyan.

Initially only the plot of the singular values and the corresponding first left and right singular vectors are shown. The user can then change the number of left and right singular vectors shown in increments of one and at each step judge the quality of the structure in each of the singular vectors. Once the singular vectors start showing too little structure or too noisy behavior this is an indication that the number of independent component has been exceeded.

## 2.2. Visual modeling

After the initial pre-processing of the data is done and a dataset is created, a first attempt can be made to model and fit these data. The best approach is often to start as simple as possible, choosing reasonable starting values for the minimum set of parameters that is expected to adequately describe the data. The chosen parameters should then be optimized by running the model for a few iterations and re-adjusting the starting values based on the results. Upon convergence of the chosen model the quality of the fit can be judged from the fitted spectra and time traces. At this point careful inspection of the SVD of the residual matrix can justify gradually increasing the complexity of the model to account for any misfit. This process of posing a model, fitting, inspecting the results and re-adjusting the model is called interactive modeling (Van Stokkum and Bal 2006). Figure 3 shows a screenshot of the analysis scheme editor where a large part of this process takes place.

The analysis scheme editor enables the user to create and edit models (as part of the analysis scheme). Figure 3 shows a screenshot of the editor together with the Palette and the Property editor. The palette is used to drag various modeling parameters onto the model view which can then be changed by selecting them and using the Property editor. For each parameter

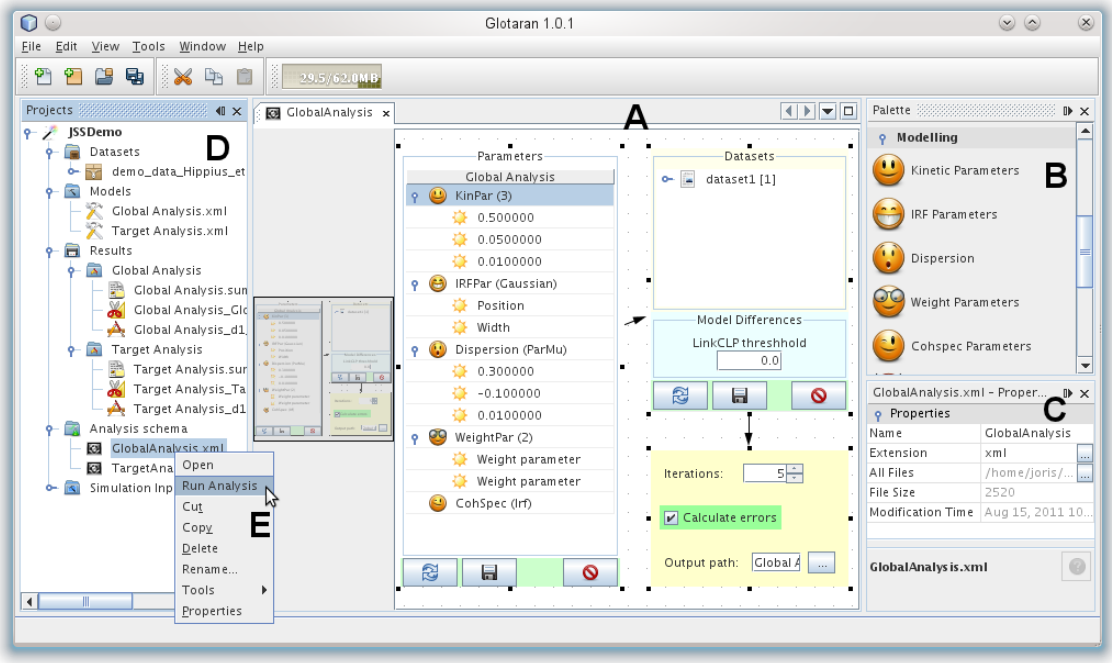


Figure 3: Screenshot of the application window under Linux with the following components: (A) analysis scheme editor showing a model for global analysis indicating all parameters $\theta$ that define the concentration $c_{l,\lambda}(t, \theta)$, (B) the corresponding palette and (C) property editor (discussed in the text). (D) shows the project folder structure with the used datasets, models, analysis scheme and resulting analysis files. (E) is the context menu for the analysis scheme file from which the analysis is invoked. The entire demo project shown in this screenshot is available from the `New Project...` item in the application menu.

selected only the properties that are relevant given the selected model are shown.

The analysis scheme editor provides an overview of the whole analysis process. A model consisting of a list of parameters is applied to one or more datasets and this is then linked to an output window. The model specification view consists of a so-called tree table view containing a list of parameters. Most parameter nodes can be expanded to view the underlying parameters, as indicated by the 'o-' symbol. The model specification is directly linked to a model specification file in the project's model directory in which the parameters of the models are stored. The `KinPar` node represents a number of kinetic rates. The `IRFPar` node represents the instrument response function (IRF). The `Dispersion` node models dispersion of the IRF in case it has a wavelength dependent character. `WeightPar` can be used to give weights to certain regions of the data indicated by two numbers specifying the interval in one or two dimensions, and one number specifying the weight to be applied there. Finally, the `CohSpec` node holds the specification for the coherent artefact; typically modeled after the IRF.

The current visual modeling implementation is rather rudimentary. Another and perhaps more intuitive way is to directly visualize the applied target scheme in the form of a visual diagram. In principle this is supported by the **NetBeans** RCP and this issue is addressed in Section 5.2 on future work.

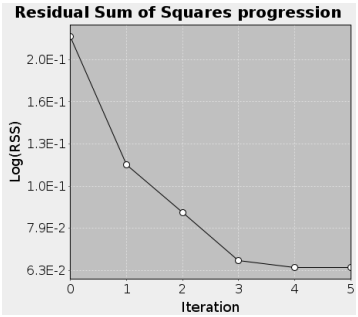


Figure 4: Progression of the sum of squares of errors. The constant value of the SSE from the 4th to the 5th iteration indicates convergence.

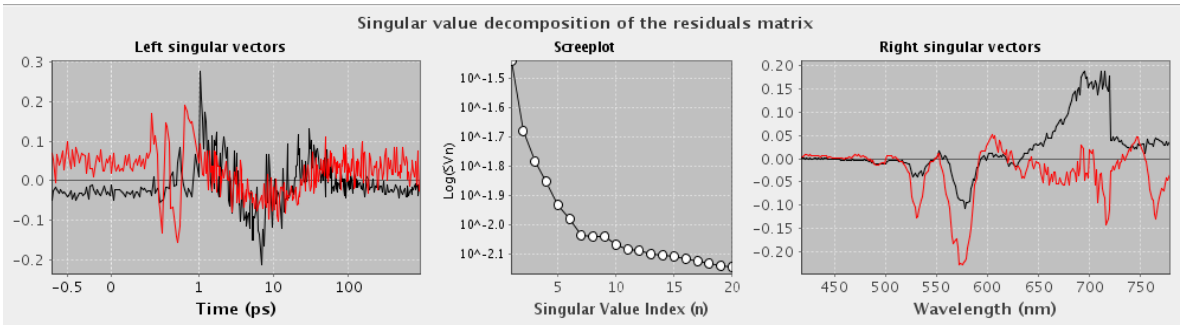


Figure 5: The singular value decomposition of the residual matrix. The first singular vector is shown in black; the second in red.

## 2.3. Parameter estimation

The above defined model for the concentration of each component $l$ depends on all parameters $\theta$ indicated in Figure 3. Thus Equation 1 can be represented mathematically:

$$\Psi(\lambda, t) = \sum_{l=1}^{n_{comp}} c_{l,\lambda}(t, \theta)\epsilon_l(\lambda) \tag{3}$$

Now that a model for the concentration of each component has been defined, the unknown parameters $\theta$ and the conditionally linear parameters $\epsilon_l(\lambda)$ can be estimated by means of variable projection (Golub and LeVeque 1979; Bates and Watts 1988; Mullen and Van Stokkum 2009). The input settings for the parameter estimation are the weight parameters and the number of iterations (both indicated in Figure 3).

Judging the quality of the fit is now a three step process. First convergence of the fit can be determined from the plot showing the progression of the sum of square of errors (SSE), see Figure 4. When the SSE has not yet converged, more iterations are needed.

Once convergence has been established, the second step is to inspect the SVD of the residual matrix. Any significant residual structure in the first two left and right singular vectors could indicate a potential misfit and the need to model another component, or that the contribution of different regions to the parameter estimation should be reweighted. The SVD of the residual matrix is shown in Figure 5.

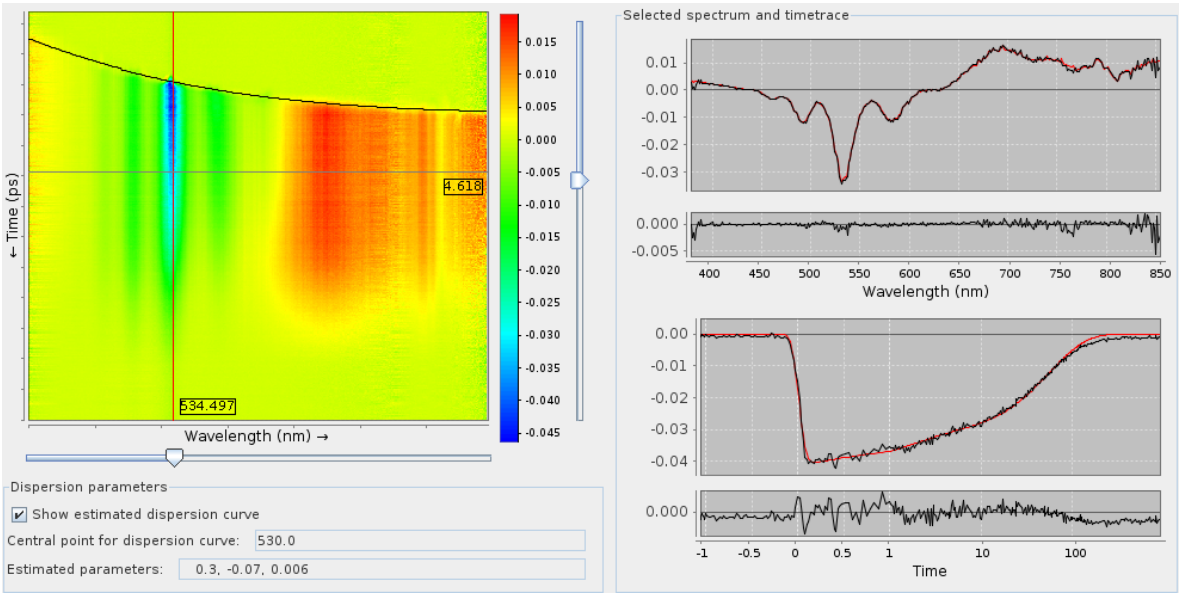


Figure 6: The fitted traces overlaid on the original data. The left panel shows the original data, overlaid with the fitted dispersion curve. The data are shown along with the fitted traces for the time point and wavelength indicated by the cross-hair on the chart.

When the SVD of the residual matrix shows no significant residual structure, the next step is to look at the results of the fit. Here it helps to first look at the original data overlaid with the fitted traces, as in Figure 6.

At this point the user can decide to accept the fit and next check the precision of the individual estimated parameters by looking at the standard errors outputted by the program. Finally, if the analysis is deemed successful, the last step in the process of interactive modeling consists of inspecting the analysis results and interpreting the resulting spectra and concentration profiles, as discussed in the next section. The overview of the fitted traces shows the original data, possibly with an overlaid dispersion curve estimating the maximum of the IRF. The scrollbars to either side of the data window can be used to inspect the fitted time traces and spectra at various locations.

## 2.4. Interactive results inspection

The results window for the analysis of time-resolved spectroscopy data consists of a total of 4 tabs. The second tab was already presented in Figure 6; it gives an idea of the quality of the fit and shows the estimated dispersion parameters (if any). In this tab the user is able to scroll through the fitted data and take snapshots of selected time traces or spectra, which are then shown together in the third and fourth tab. The first tab is the most important because here the results of the analysis are summarized. As an example the results of the target analysis of the data presented in Figure 1 are shown in Figure 7.

In Figure 7 are shown: The SVD of the complete residual matrix (bottom row), the estimated kinetic parameters (top right), the spectra (SAS or EAS and DAS), the normalized spectra, and the concentration profiles. The mathematical models underlying the various spectra are discussed below.

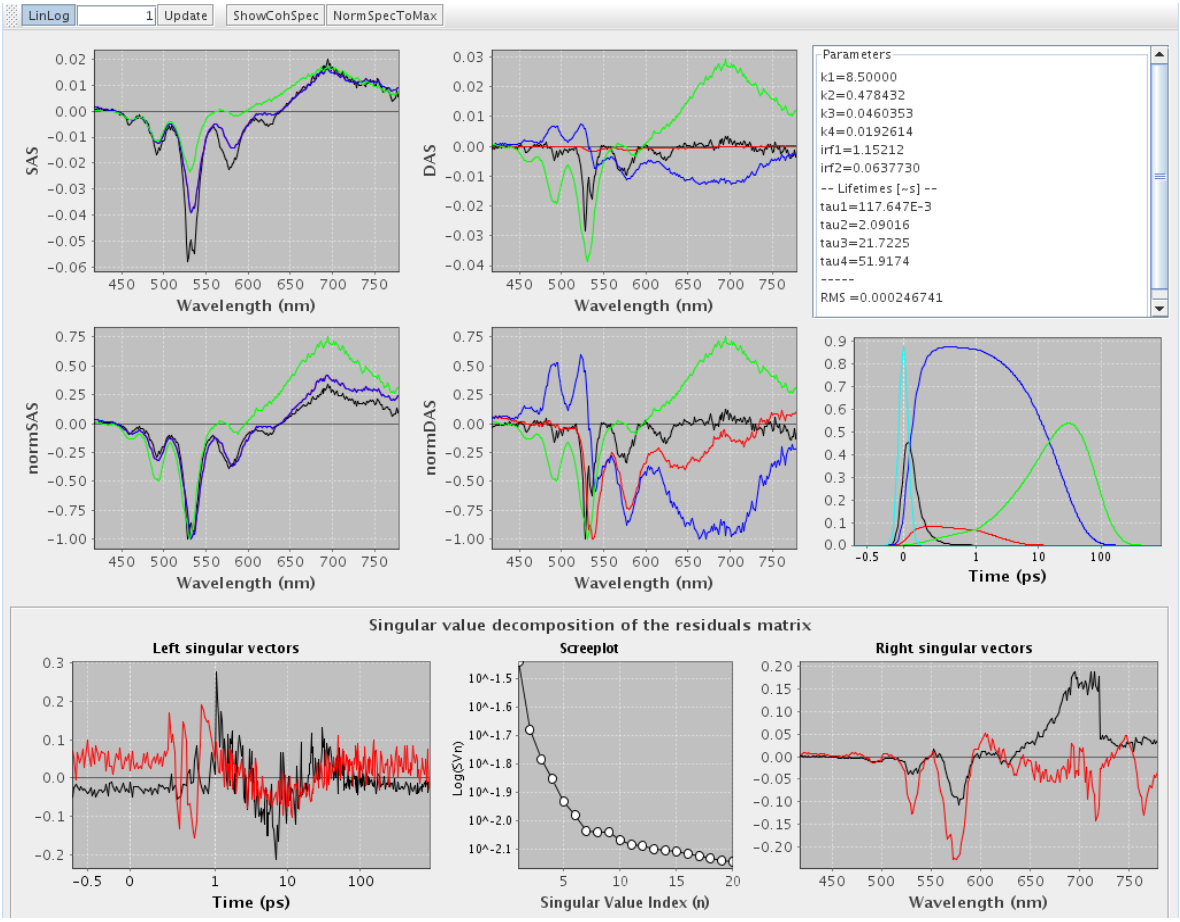


Figure 7: Overview tab showing the results of target analysis (from Hippius *et al.* 2007, Figure 6 and 7). The top row shows the estimated SAS (see Equation 6), the DAS (see Equation 4), which are computed from the SAS, and the estimated parameters. The middle row displays the normalized spectra and the concentration profiles of the components. Here black represents the unrelaxed excited state, red and blue represent the relaxed excited state decaying fast and slow respectively, green is the charge transfer state and finally cyan represents the coherent artifact. The bottom row shows the SVD of the residual matrix.

DAS are decay associated spectra, and the model of Equation 1 then reads

$$\Psi(\lambda, t) = \sum_{l=1}^{n_{comp}} c_l^{\mathrm{DAS}}(t, \theta)\mathrm{DAS}_l(\lambda) \tag{4}$$

where $c_l^{\mathrm{DAS}}(t, \theta)$ is the exponentially decaying concentration of component $l$ convolved with the IRF.

More complicated concentrations are linear combinations of exponential decays, which are again $c_l^{\mathrm{DAS}}(t, \theta)$. They are often described as compartmental models (Godfrey 1983; Bates and Watts 1988). In a sequential or unbranched unidirectional model the associated spectra

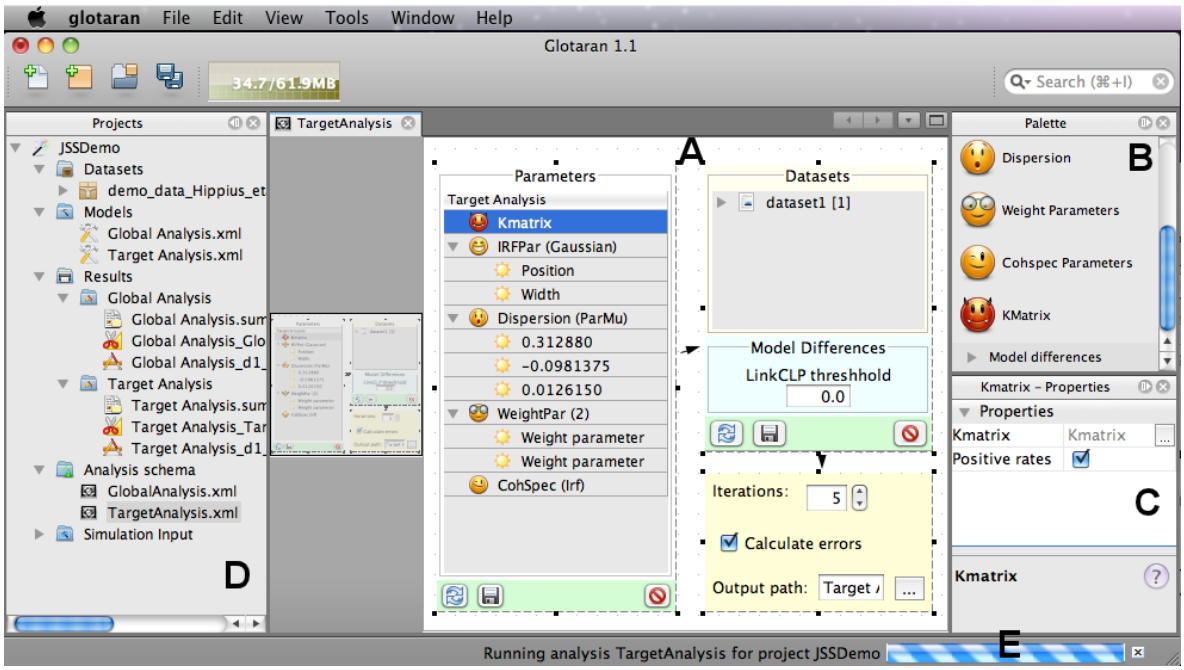


Figure 8: Screenshot of the application window under Mac OS X displaying the (A) analysis scheme editor showing a model for target analysis, (B) the corresponding palette and (C) the property editor, (D) the project folder structure with the used datasets, models, analysis scheme and resulting analysis files and (E) the progress bar showing the current running analysis. Note that here the kinetic parameters are specified in the form of a `Kmatrix` node, which opens in an external editor shown in Figure 9. The entire demo project shown in this screenshot is available from the `New Project...` item in the application menu.

are called evolution associated spectra (EAS), and the model of Equation 1 then reads

$$\Psi(\lambda, t) = \sum_{l=1}^{n_{comp}} c_l^{\mathrm{EAS}}(t, \theta)\mathrm{EAS}_l(\lambda) \tag{5}$$

With successively increasing lifetimes (an example is depicted in Figure 13) the EAS visualize the spectral evolution.

A compartmental model inspired by scientific hypotheses and assumptions is termed a target model, and this type of global analysis is referred to as target analysis. With a target model the associated spectra are called species associated spectra (SAS), and the model of Equation 1 then reads

$$\Psi(\lambda, t) = \sum_{l=1}^{n_{comp}} c_l^{\mathrm{SAS}}(t, \theta)\mathrm{SAS}_l(\lambda) \tag{6}$$

Both $c_l^{\mathrm{EAS}}(t, \theta)$ and $c_l^{\mathrm{SAS}}(t, \theta)$ are linear combinations of all $c_l^{\mathrm{DAS}}(t, \theta)$. Thus from the EAS or SAS new DAS can be computed, which obey the assumptions of the kinetic model used.

The top row of the overview tab of the results shows the estimated EAS (or SAS), the DAS computed from the EAS (or SAS), and the estimated parameters. When a sequential kinetic model is used, the estimated parameters are the lifetimes of the components; when a target

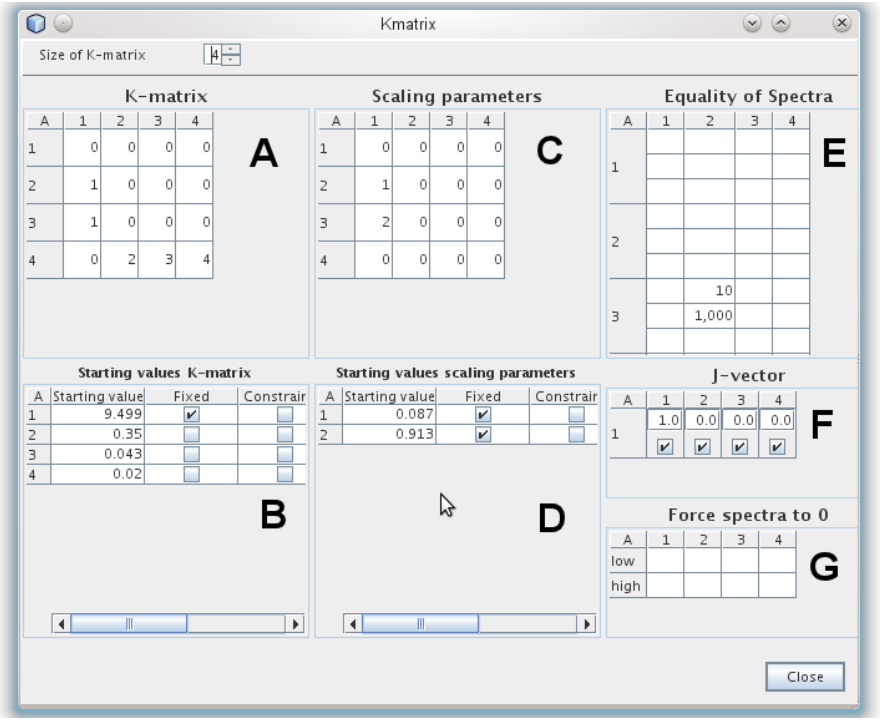


Figure 9: Screenshot of the `Kmatrix` editor. Here (A) is the matrix that describes the spectral evolution, the rows and columns represent the different compartments, a non-zero value in a particular cell indicates transfer from the column to the row compartment. For example, in this case compartment 1 transfers into compartment 2 and 3. The starting values for the rates of transfer are specified in (B), the index corresponds to the number in (A). In the case of branching (one compartment transfers to 2 or more different compartments) so called scaling parameters can be defined in (C) with the starting values specified in (D) which indicates the ratio of branching. In this case compartment 1 transfers with a rate of 8.7% of the total rate to compartment 2 and 91.3% of the total rate to compartment 3. In (E) spectral equality constraints can be specified, here the spectrum of compartment 2 is made equal to the spectrum of compartment 3 from 10 to 1000 units (nm). In (F) the initial population of the compartments (directly after the moment of excitation) is given. In this case only the first compartment is initially populated. Finally in (G) it is possible to force parts of the spectrum of a compartment to zero. The lower and upper wavelength can be specified. This whole K-matrix is schematically depicted in Figure 11.

kinetic model is used, these are the microscopic rate constants. For easier comparison the normalized spectra are also displayed. A summary of the estimated parameters is shown in a scrollable text field. The concentration profiles of the components are displayed below that. If a coherent artifact was modeled, it will be shown here, and can also be shown in the EAS (or SAS) by toggling the `ShowCohSpec` in the toolbar of the window (not shown here). Finally the SVD of the residual matrix is again displayed to inspect the quality of the fit and help determine if there are any trends left in the residuals that might need to be modeled. For example, severe misfit of the dynamics of a particular component can cause the shape of the first right singular vector to resemble the SAS of that component.

The results of the target analysis of the data from Figure 1 have already been presented in Figure 7. The GUI screenshots showing the model specification for this analysis are shown in Figure 8 and Figure 9.

## 2.5. Comparison with TIMP

To provide an idea of the difference between working with the graphical user interface in **Glotaran** versus the command line interface in R, the step-by-step commands to reproduce the above target analysis with **TIMP** are given below. In **Glotaran** the `readData` and `preProcess` commands are not used because data handling and preprocessing is taken care of by the GUI but they are included here for completeness' sake. The `initModel` and `fitModel` commands are automatically generated by **Glotaran** (and typically hidden from the user). First load the **TIMP** package into memory.

```
R> library("TIMP")
```

Second, create a dataset by reading in the demo data file using the `readData` function. Note that this data file can be obtained from the `resources` folder of a Demo project generated using **Glotaran**.

```
R> dataset <- readData(
+    "demo_data_Hippius_etal_JPCC2007_111_13988_Figs5_9.ascii")
```

Third, select a window from the data to analyze using the `preProcess` function. Here the entire file is selected because the demo dataset has already been pre-processed.

```
R> dataset <- preProcess(data = dataset, sel_time = c(1, 335))
```

Fourth, initialize the model using the function `initModel`. This includes specifying values for the following arguments: `kinpar`, `kmat`, `kinscal` and `jvec` (detailing the (kinetic) model to be used for analysis), `clpequspec` and `clpequ` (specifying what spectral constraints should be used), `irfpar`, `parmu`, `lamdac` and `dispmufun` (specifying the parameters related to the IRF and the dispersion thereof), `cohspec` (detailing the type of coherent artifact to be included in the model) and finally `fixed` (specifying which parameters should remain fixed during optimization).

```
R> model <- initModel(mod_type = "kin", kinpar = c(9.5, 0.35, 0.043, 0.02),
+    kmat = array(c(0, 1, 1, 0, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0, 4, 0, 1, 2,
+      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), dim = c(4, 4, 2)),
+    jvec = c(1.0, 0.0, 0.0, 0.0), kinscal = c(0.087, 0.913),
+    irfpar = c(1.2, 0.05), lambdac = 550.0,
+    parmu = list(c(0.3, -0.1, 0.01)), dispmufun = "poly",
+    positivepar = c("kinpar"), cohspec = list(type = "irf"),
+    clpequspec = list(list(to = 3, from = 2, low = 10.0, high = 1000.0)),
+    clpequ = 1, weightpar = list(c(NA, NA, 280.0, 550.0, 0.1),
+      c(NA, NA, 720.0, 890.0, 0.2)), fixed = list(kinpar = 1,
+      kinscal = c(1, 2), jvec = c(1, 2, 3, 4), clpequ = 1:1),
+    seqmod = TRUE)
```

Finally, to fit the model to the data and plot the results the function `fitModel` should be called.

```
R> gtaFitResult <- fitModel(data = list(dataset),
+    modspec = list(model), modeldiffs = list(linkclp = list(1)),
+    opt = kinopt(iter = 5, stderrclp = TRUE, kinspecerr = TRUE, plot = TRUE,
+      linrange = 1, xlab = "Time (ps)", ylab = "Wavelength (nm)",
+      plotkinspec = TRUE, selectedtraces = seq(1, dataset@nl, by = 10)))
```

The complexity of the commands above shows the advantage of using a GUI: Typographical mistakes are eliminated and model specification is simplified. For instance, compare the specification of the K-matrix in **TIMP** (the `kmat` variable in the commands above) and the model specification in **Glotaran** (see Figure 9). In **TIMP** the K-matrix and the scaling parameters are specified in a single array. which is hard to read and prone to typographical errors. In **Glotaran**, these matrices are shown separately and correctly formatted, facilitating specification of the correct model.

## 3. Requirements and design choices

The primary user requirements for **Glotaran** are determined by the functionality and limitations of the package **TIMP** for which it is intended to be the graphical front end. Ideally, **Glotaran** would allow access to all of **TIMP**'s functionality while extending it with new and useful features. Currently most of the functionality of **TIMP** is accessible from **Glotaran**. Datasets and models created in **Glotaran** are translated into data and models that can be used in **TIMP**. Some additional requirements, based on practical considerations and feedback on initial versions are summarized below.

- The application must be open-source and platform-independent.
- It should implement (most of) the functionality already provided by **TIMP**; partially this is achieved by using **TIMP** as the computational back-end for analysis.
- It should be designed to facilitate interactive data-exploration, without any scripting or code manipulation on the user's end.
- It should support a simple, preferably visual, model specification that uses an intuitive graphical user interface, rather than a language based (syntax-sensitive) model specification.
- Just as it supports interactive data-exploration, it should support interactive inspection of the results (scrolling through the fitted data, zooming in on results, comparing results from multiple different models, etc.)
- Development of the application should be done using a collaborative IDE, with support for visual GUI design. In addition the IDE used should also be cross-platform and open-source, eliminating any barriers for potential developers.

- In order to minimize the time spent on so-called 'plumbing' code[1], the application should be implemented on top of a RCP. In an academic environment with very limited resources (often just a single programmer) and limited time this is a very important requirement. The use of a RCP also adds much desired modularity which facilitates maintenance and extensibility by future developers.

Various combinations of programming languages, IDE's and external libraries were evaluated against these requirements. We ultimately selected a Java-based desktop application on top of the **NetBeans** RCP, using the **NetBeans** IDE as a development environment, making extensive use of the IDE's excellent Swing GUI builder and seamless integration with the free project hosting site http://kenai.com/[2] for collaborative development. The charting library **JFreeChart** was chosen to visualize all plots. **JFreeChart** is a very well documented open-source chart library for Java with many example implementations available through the **JFreeChart** developer's guide and the forums (Gilbert 2011). The library has support for interactivity in the form of clicking, zooming and panning and exporting to various graphics formats is possible. The ability to actually create an application in Java which can still communicate with R is provided by the open-source communication library **Rserve**. **Rserve** is a TCP/IP server which allows other programs to use facilities of R from various languages – including Java – without the need to initialize R or link to the R library (Urbanek 2003).

## 4. Application architecture

The **Glotaran** application is completely written in the programming language Java and is being developed on top of the **NetBeans** platform using the **NetBeans** IDE. It relies on several external libraries for plotting, data-handling and communication with the R computing environment. **Glotaran** is completely open-source and the source code is published under the terms of the GNU general public license. It is cross-platform and runs under any operating system that has a recent version of the Java virtual machine installed. Figure 10 shows the application architecture diagram.

The application diagram shows **Glotaran** running on top of the **NetBeans** RCP, which sits on top of the Java virtual machine (JVM) controlled by the operating system (OS) of the client. The **Rserve** library facilitates the communication with the **Rserve** binary via the TCP/IP protocol. The **Rserve** binary is launched by the R computing platform controlled by the server's OS. The client and the server can be physically on the same machine, allowing for the fastest communication between Java and R. However, the server can also be a much faster compute server located somewhere in the network (or the Internet), allowing for much faster processing (at the price of slower communication). In addition the client would not need to have R installed. In the following section the various blocks in the application diagram are discussed further.

**Glotaran** is not a traditional stand-alone Java application, it is actually a collection of modules that integrates into the **NetBeans** RCP. The **NetBeans** platform is a reusable framework for

[1]The 'plumbing' of an application refers to the basic functionality that almost any application needs, such as: Application state saving, connecting actions to menu items, toolbar items and keyboard shortcuts, window management (docking, resizing), and so on. An RCP provides all of this very basic functionality out of the box, freeing time to work on the application's higher-level functionality.

[2]http://kenai.com/ will be superseded by http://java.net/ by the end of 2012 according to its project website.

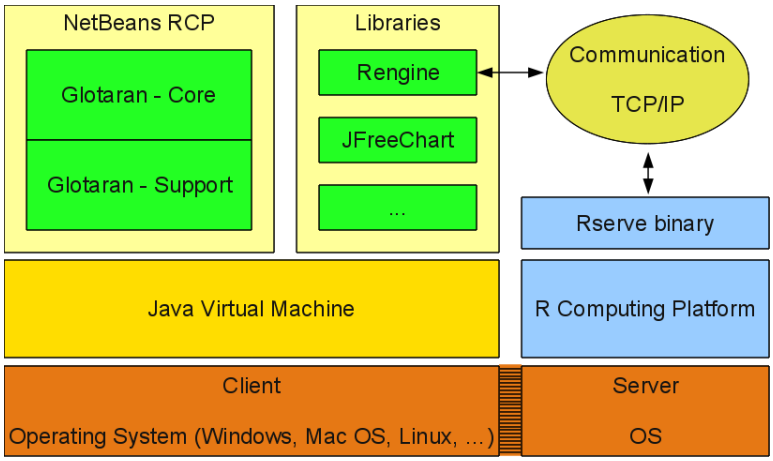


Figure 10: Application diagram showing the interaction between **Glotaran**, R and Java.

simplifying the development of Java Swing desktop applications. The platform offers reusable services common to desktop applications, allowing developers to focus on the logic specific to their application rather than the basic functionality that is part of any desktop application (i.e., interface management, windows management, storage management, etc).

The core of **Glotaran** consists of the modules that provide the user interface components and handle user input. This includes the data visualization, the visual modeling, the result visualization and the analysis worker (the module that is responsible for interpreting the user's input and delegate it to the available computational support module). The idea behind separating interpretation of the module and the actual computation is interchangeability. Currently, all computation is done within the R package **TIMP**, but it could be done using any software package as long as the interface with **Glotaran** is clearly defined in a computational support module.

## 4.1. Libraries

**Glotaran** depends on various external libraries for plotting, communicating with the R computing platform, reading in various file formats and to provide for various mathematical subroutines.

### *JFreeChart*

**JFreeChart** is the main plotting library used in **Glotaran**. It has support for many output types, including Swing components (on display graphics), image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG). A wide variety of standard plots is available from within the library[3], and **JFreeChart** can be extended to support different types of plots as well. For this project **JFreeChart** was extended to support a new type of Axis called LinLogAxis, as detailed in Appendix A. **JFreeChart** is being developed by David Gilbert and the **JFreeChart** community members.

[3]Examples of the use of **JFreeChart** can be found at: http://www.jfree.org/jfreechart/samples.html.

#### *Rserve*

REngine is the library provided with the **Rserve** software package, to facilitate the communication between Java and the R server application provided by **Rserve**. As a wrapper library for REngine the library JRConnect is used. This ensures stability over multiple iterations and generations of REngine. JRConnect was developed by Ralf Seger.

## 5. Conclusions

**Glotaran** has been introduced as an open-source, cross-platform Java based graphical user interface for the R Package **TIMP** to facilitate the process of interactive global and target data analysis. In comparison with existing software **Glotaran** adds: Interactive data-visualization, visual modeling and interactive inspection of analysis results. Its well defined 'project' structure facilitates collaborative analysis. Initially its use is mostly targeted towards applications in the field of Biophysics, specifically for the analysis of time-resolved spectroscopy and microscopy data analysis. By making use of the **NetBeans** RCP it was possible to assemble a highly modular and extensible application. Despite being fully written in Java, it can still harness the power of R through the rJava interface REngine. R here is used as a computational engine rather than a full fledged programming environment, delegating all but the core computational operations to proglangJava subroutines instead.

### 5.1. Availability

As mentioned before, **Glotaran** is an open-source application, and is available under the terms of the GPL license. As such, the application's source code can be checked out by interested developers from the project's hosting site at http://kenai.com/projects/glotaran/. However, for most users the binary installers, available for the platform Windows, GNU Linux and Mac OS X, are more practical and can be acquired through the project's main website: http://glotaran.org/. Installation instruction, general documentation and various screenshots and video demonstrations are also available.

### 5.2. Future work

*Visual compartmental modeling*

Compartmental kinetic models are represented in **Glotaran** by a combination of a transfer matrix K, a vector representing the kinetic parameters (the rate constants), a vector representing the branching parameters and finally a vector of inputs (representing the IRF), see Figure 9. However, it would be advantageous to represent a compartmental model as a target kinetic scheme, since this is typically how biophysicists think. In **Glotaran** this could be implemented using the **NetBeans** Visual Library (the same system in place to design the analysis scheme) which is part of the **NetBeans** RCP.

*Reporting functionality*

iReport is the free, open source report designer for JasperReports. It allows the user to create very sophisticated layouts containing charts, images, subreports, crosstabs and much more.

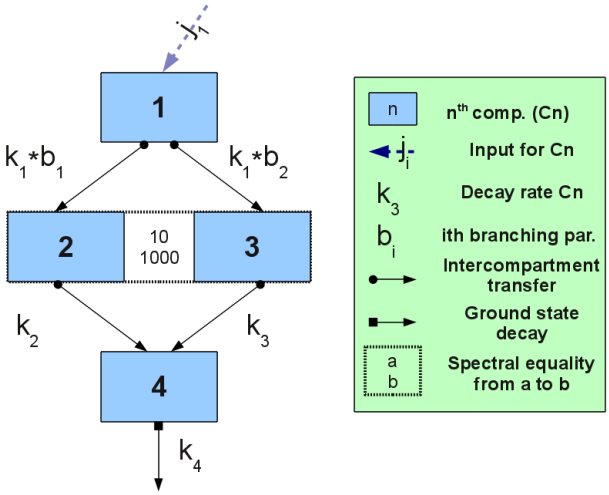


Figure 11: Target model (from Hippius *et al.* 2007, Figure 7): 1 unrelaxed excited state, 2 and 3 relaxed excited state decaying fast or slowly (identical SAS), 4 charge transfer state.

Data can be accessed through a wide variety of methods and support for **Glotaran** as a data source could be implemented. It would then be possible to compose an analysis results report and publish it as PDF, RTF, XML, XLS, CSV, HTML, XHTML, text, DOCX, or OpenOffice.

*Integration with databases*

One of the development criteria that was put forward for the development **TIMP** was a graphical user interface supporting collaborative research and enabling distributed interactive modeling, where an expert in modeling and parameter estimation can analyze the data and an experimental scientist can contribute to the interactive modeling by discarding unrealistic models and suggesting model improvements. The first step for this was made in **Glotaran** by introducing the system of project-management, e.g., a project - consisting of datasets, models and analysis schema can easily be 'zipped up' and sent to a collaborator. The next step is the integration of **Glotaran** with a database system for easy models storage and potentially the storage of (selected) measurements as well. The **NetBeans** platform already offers the integration with databases 'for free' so all that is left is the implementation within **Glotaran**.

## Acknowledgments

This research was funded by Computational Science grant #635.000.014 from the Netherlands Organization for Scientific Research (NWO). Catharina Hippius and René Williams provided the data modeled in Section 2. We acknowledge the constructive criticism of the anonymous reviewers.

## References

Bates DM, Watts DG (1988). *Nonlinear Regression Analysis and Its Applications*. John Wiley & Sons, New York.

Böck H (2011). *The Definitive Guide to **NetBeans** Platform 7*. Apress.

Fellows I (2012). "**Deducer**: A Data Analysis GUI for R." *Journal of Statistical Software*, **49**(8), 1–15. URL http://www.jstatsoft.org/v49/i08/.

Gilbert D (2011). "**JFreeChart**." http://www.jfree.org/jfreechart/.

Godfrey K (1983). *Compartmental Models and their Application*. Academic Press, London.

Golub GH, LeVeque RJ (1979). "Extensions and Uses of the Variable Projection Algorithm for Solving Nonlinear Least Squares Problems." In *Proceedings of the 1979 Army Numerical Analysis and Computers Conference*, volume ARO Report 79-3, pp. 1–12.

Golub GH, Van Loan CF (1996). *Matrix Computations*. 3rd edition. The Johns Hopkins University Press, Baltimore.

Grosjean P (2010). "R GUI Projects." URL http://www.R-project.org/GUI.

Helbig M, Theus M (2005). "**JGR**: Java GUI for R." *Statistical Computing and Graphics Newsletter*, **16**(2), 9–12. URL http://stat-computing.org/newsletter/issues/scgn-16-2.pdf.

Hippius C, Van Stokkum IHM, Zangrando E, Williams RM, Würthner F (2007). "Excited State Interactions in Calix[4]arene-Perylene Bisimide Dye Conjugates: Global and Target Analysis of Supramolecular Building Blocks." *Journal of Physical Chemistry C*, **111**(37), 13988–13996.

Laptenok S, Borst JW, Mullen KM, Van Stokkum IHM, Visser AJWG, Van Amerongen H (2010). "Global Analysis of Förster Resonance Energy Transfer in Live Cells Measured by Fluorescence Lifetime Imaging Microscopy Exploiting the Rise Time of Acceptor Fluorescence." *Physical Chemistry Chemical Physics*, **12**, 7593–7602.

Laptenok S, Mullen KM, Borst JW, Van Stokkum IHM, Apanasovich VV, Visser AJWG (2007). "Fluorescence Lifetime Imaging Microscopy (FLIM) Data Analysis with **TIMP**." *Journal of Statistical Software*, **18**(8), 1–20. URL http://www.jstatsoft.org/v18/i08/.

Mullen KM, Van Stokkum IHM (2007). "**TIMP**: An R Package for Modeling Multi-Way Spectroscopic Measurements." *Journal of Statistical Software*, **18**(3), 1–46. URL http://www.jstatsoft.org/v18/i03/.

Mullen KM, Van Stokkum IHM (2009). "The Variable Projection Algorithm in Time-Resolved Spectroscopy, Microscopy and Mass Spectrometry Applications." *Numerical Algorithms*, **51**, 319–340.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Rödiger S, Friedrichsmeier T, Kapat P, Michalke M (2012). "**RKWard**: A Comprehensive Graphical User Interface and Integrated Development Environment for Statistical Analysis with R." *Journal of Statistical Software*, **49**(9), 1–34. URL http://www.jstatsoft.org/v49/i09/.

Snellenburg JJ, Van Stokkum IHM, Mullen KM (2008). "TIMPGUI: A Graphical User Interface for the Package **TIMP**." Talk at useR! 2008, The R User Conference (Dortmund, Germany), URL http://www.statistik.uni-dortmund.de/useR-2008/slides/Snellenburg+Mullen+van_Stokkum.pdf.

Urbanek S (2003). "**Rserve** – A Fast Way to Provide R Functionality to Applications." In K Hornik, F Leisch, A Zeileis (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*. Vienna, Austria. ISSN 1609-395X. URL http://www.ci.tuwien.ac.at/Conferences/DSC-2003/.

Valero-Mora PM, Ledesma R (2012). "Graphical User Interfaces for R." *Journal of Statistical Software*, **49**(1), 1–8. URL http://www.jstatsoft.org/v49/i01/.

Van Stokkum IHM, Bal HE (2006). "A Problem Solving Environment for Interactive Modelling of Multiway Data." *Concurrency and Computation: Practice and Experience*, **18**, 263–269.

Van Stokkum IHM, Larsen DS, Van Grondelle R (2004). "Global and Target Analysis of Time-Resolved Spectra." *Biochimica et Biophysica Acta*, **1657**, 82–104. Erratum in **1658**, p. 262.

# A. Extending JFreeChart

**JFreeChart** is an extensive library with support for a great number of different types of plots. The project website gives a list of examples. However, one type of plot frequently used in the visualization of time-resolved spectroscopy data, the linear-logarithmic plot, is not included in the standard set of plots. Because this type of plot is essential for the working of a global and target analysis tool, it was developed as part of the project. The result is the Java class `LinLogAxis` and implementation of the standard **JFreeChart** class `org.jfree.chart.axis.ValueAxis`. Because the LinLog functionality was implemented as a ValueAxis, it integrates seamlessly with the standard **JFreeChart** chart library and can be reused with any XY Plot. The improvement for certain types of data is impressive when compared side by side; see Figure 12.

The left plot in Figure 12 uses the standard linear `NumberAxis` only, and the right plot uses the new LinLog axis. Figure 13 displays the estimated concentration profiles of a sequential model with lifetimes ranging from 1.76 till 54 ps. The additional information provided by the `LinLogAxis` is obvious.

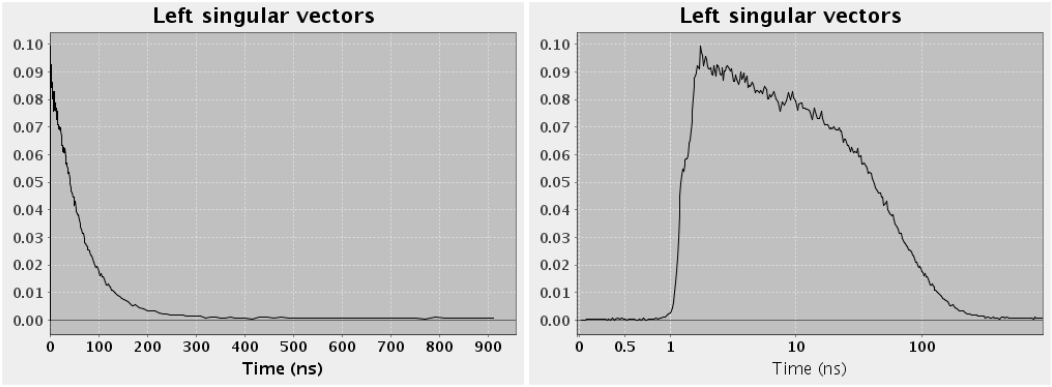


Figure 12: Example of the use of the `LinLogAxis` implementation.

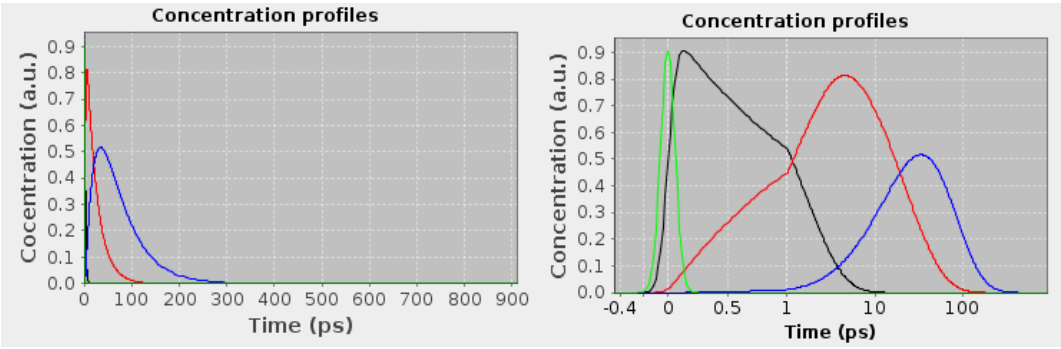


Figure 13: Two charts showing the estimated concentration profiles of a sequential model, the left chart uses the standard linear axis and the right charts uses the linear-logarithmic axis. Lifetimes: 1.76 ps (black), 18.7 ps (red), and 54 ps (blue). Green represents the shape of the coherent artifact, and is equal to the IRF.

**Affiliation:**

Joris J. Snellenburg, Ivo H. M. van Stokkum
Department of Physics and Astronomy, Faculty of Sciences
VU University Amsterdam
De Boelelaan 1081
1081 HV Amsterdam, The Netherlands
E-mail: j.snellenburg@vu.nl, ivo@few.vu.nl
URL: http://www.few.vu.nl/~jsnel/, http://www.few.vu.nl/~ivo/

Sergey P. Laptenok
Laboratoire d'Optique et Biosciences
CNRS UMR 7645, INSERM U696
L'École Polytechnique
F-91128 Palaiseau, France
E-mail: s.laptenok@vu.nl
URL: http://www.lob.polytechnique.fr

Ralf Seger
Department of Computer Oriented Statistics and Data Analysis (COSADA)
Augsburg University
D-86135 Augsburg, Germany
E-mail: ralfseger@googlemail.com
URL: http://rosuda.org/Moret/main.html

Katharine M. Mullen
Structure Determination Methods Group, Ceramics Division
National Institute of Standards and Technology (NIST)
100 Bureau Drive, M/S 8520
Gaithersburg, MD, 20899, United States of America
E-mail: mullenkate@gmail.com

*Journal of Statistical Software* http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 49, Issue 3 *Submitted:* 2011-01-20
June 2012 *Accepted:* 2011-09-16