A Versatile Scientific Programming Environment for Cardiovascular Data Analysis

A. Schlemmer, T. Hajek, U. Parlitz, S.E. Lehnart, N. Wessel, H. Malberg and S. Luther

***Abstract*—The analysis of complex cardiovascular signals is a rapidly emerging field in biomedical engineering with significant potential for the development of clinical diagnostics and therapy. The development of signal processing algorithms and their subsequent application within large-scale cross-disciplinary research networks is a particular challenge. We have developed a novel software tool for multi-platform scientific programming, biomedical data analysis and visualization for collaborative, interdisciplinary research. The software is written in Java and has a graphical user interface, which provides intuitive access for non-programmers. The easy-to-use plug-in system allows scientists to extended the program's functionality as needed. This versatile software design makes the program particularly useful for cross-disciplinary and multi-level biomedical research. The software has been primarily developed for the analysis of high-resolution spatial-temporal experimental data obtained in electrophysiological optical mapping experiments in the large-scale consortium EUTrigTreat (http://www.eutrigtreat.eu/). Its functionality comprises algorithms and interactive tools for multivariate time series analysis including smoothing filters, distinct normalization methods, phase singularity analysis, activation map analysis, frequency analysis, and correlation functions. An inbuilt automatic logging mechanism documents each step of the analysis to ensure reproducibility. The software provides a generic open access framework for cardiovascular data analysis and visualization. A database with electrophysiological data is provided online for algorithm development and benchmarking. The software is available under GPL license from http://www.bmp.ds.mpg.de/software.html.**

***Index Terms*—signal analysis, java, open source software, time series, image processing, graphical user interface, video data analysis**

I. INTRODUCTION

Analysis of cardiovascular data often involves the application of high level algorithms. On the other hand major user interactions are needed during the process. A new open source analysis environment called OpenAnalyser tries to close the gap between an easy-to-use graphical user interface and a flexible development platform for integration and adaption of new algorithms. Furthermore we want to provide a platform for development, standardization and exchange of algorithms and establish methods for transparent access to algorithms and data used in the analysis of cardiovascular data. An automatic logging mechanism helps preserving the overview about the analysis process. A database connection is in preparation for instant access to data and results via sophisticated search methods.

A. Schlemmer, T. Hajek and S. Luther are with the Max Planck Research Group Biomedical Physics, Max Planck Institute for Dynamics and Self-Organization, 37077 Göttingen, Germany, e-mail: alexander.schlemmer@ds.mpg.de

U. Parlitz is with Drittes Physikalisches Institut, Georg-August-Universität Göttingen, 37077 Göttingen, Germany

S.E. Lehnart is with the Depts. of Cardiology & Pulmonology and Pharmacology, Heart Research Center Goettingen (HRCG), University Medical Center (UMG), Robert-Koch-Str. 40, 37075 Göttingen, Germany

N. Wessel is with the AG Nichtlineare Dynamik (S) / Kardiovaskuläre Physik, Institut für Physik, Humboldt-Universität zu Berlin, Robert-Koch-Platz 4, 10115 Berlin, Germany

H. Malberg is with the Institut für Biomedizinische Technik, Technische Universität Dresden, Fakultät Elektrotechnik und Informationstechnik, Helmholtzstraße 18, 01069 Dresden, Germany

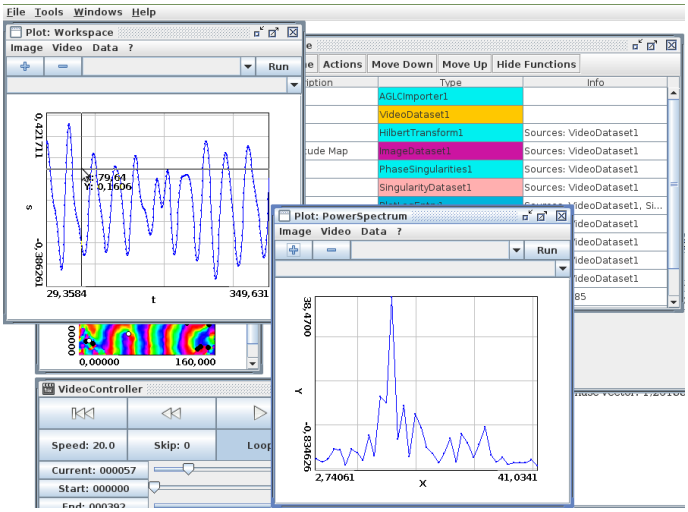

Fig. 1. Screenshot of OpenAnalyser during video data analysis.

A. Comparison to other projects

There exists a lot of software for scientific data analysis (see e.g. [1–11]). However, most open source software packages require programming knowledge or are restricted to a rather small field of data analysis.

We are aware that there are already projects pointing in similar directions, for example ImageJ, Biosig and Physionet, each being focused on different fields of interest:

- ImageJ [10] is a high level scientific image processing and analysis program.
- Biosig [8; 11] provides a comprehensive library for viewing and analysing biosignals. It supports many data formats used in this field. Many routines for non-interactive data analysis are available for different platforms and different programming languages.
- Physionet [2] is an open access platform for physiologic data and analysis software. Several specialized software for time series analysis, mostly unix command line tools, is provided and documented.

However, there are several features which make OpenAnalyser a unique tool for data analysis:

- A ready-to-use graphical user interface for scientists with little computer knowledge.
- A very simple, yet powerful and flexible plugin infrastructure which allows development and modification of algorithms even during the ongoing analysis processes.
- An automatic logging mechanism which records each step of analysis and makes results transparent and reproducable. This involves a sophisticated file format for storing project-structures and dependencies in an efficient and transparent way. The file-format can be mapped into a database for fast lookup of data and information about analysis processes.
- The software is programmed in Java and therefore platform-independent, fast [12] and secure.

B. Java as a platform for scientific computing

Java is not well-established in the field of scientific computing, but has gained increasing importance recently. A discussion on this topic can be found at [13]. The most important reasons for choosing Java for our program are the following:

- Platform independence: Java programs can be run under most modern operating systems without further adaptation.
- Powerful API: Java is equipped with a very comprehensive API. A powerful GUI-toolkit is included in the java platform.
- Prevalence: The Java language is very easy to learn and therefore taught at many schools and universities. Furthermore its syntax is very similar to C / C++ which makes it understandable by a large community.
- Robustness: The language design and features of the virtual machine reduce errors and unwanted behaviour.
- Modularity and dynamic features: Existing software systems can be easily extended, for example by plugins. Parts of the software can be reloaded at runtime. There are bytecode-compilers not only for java but also for other programming languages [14], which open the possibility to extend the program with existing code written in other programming languages (like Python or Ruby [15; 16]).

II. The Program

This section gives an overview of the main features of OpenAnalyser. More information on details of the program is provided in the online manual [17]. OpenAnalyser is available under GPL license and can be downloaded from http://www.bmp.ds.mpg.de/software.html.

A. The Workspace

The main challenge in the program design was to present raw data, results and intermediate steps in an interactive graphical user interface. The interface had to be intuitive and flexible enough to allow high level operations to be applied to collections of different kinds of data.

This central role of the program is assigned to the workspace-window which may contain a list of datasets and log-entries. By using this window the user is capable of interacting with the data, e.g. visualization and the application of functions.

B. Function-Plugins

Functions are the second important part of the program. The program is equipped with a plugin loader which scans a dedicated directory for function-plugins and presents them in a popup-menu to the user. Function-plugins are simple Java class files, which contain the actual algorithms and information about the functionality of the function (e.g. parameter-documentation). This information is provided in a way that it is available to both the programmer developing the algorithm and the user.

C. Visualization

One of the most important capabilities of data analysis software is the visualization of various kinds of data. OpenAnalyser contains methods for visualization of xy-(scatter- and line-)plots, histograms, images, 2D-data-plots, and vector-plots. Furthermore it is equipped with a video-player that is able to display 3D-/video-data. For the analysis of excitable media methods for displaying complex 3D-data and trajectories (heavily used for phase singularity analysis, see IV-B) have been included.

Plots and visualizations can be combined to form documents or to view multiple videos simultaneously. Visualizers for own datatypes can be created with special types of plugins.

D. File formats

There exist several file formats for scientific data and only very few standard file formats. Most data acquisition software comes with its own file format. Often it is even necessary to deal with multiple files simultaneously, because binary data can be spread over multiple files and meta data is stored in separate files. Furthermore the representation of data does not always correspond to the file format and sometimes it might be necessary to convert the input data into a different representation, e.g. the concatenation of image files to videos or vice versa. To simplify the process of data import, OpenAnalyser implements a clear concept based on import-plugins, which are responsable for data input and integrate seamlessly into the graphical user interface of the main program.

E. Workflow

The workflow of OpenAnalyser is different to most analysis software: It can be described as a mixture between an interactive programming environment (like MATLAB, The MathWorks, Inc.) and image manipulation software like ImageJ. In a normal session the user starts importing raw-data to the workspace. The analysis process consists of consequent operations including filtering, visualization and the application of analysis algorithms.

III. Technical Details

The plugin system has been designed with a focus on simplicity and flexibility to allow scientists to quickly integrate new algorithms and test them interactively. While many available plugin systems in other programs require a lot of programming expertise our approach consequently

disclaims features unnecessary for scientific purposes and implements some efficient tools that may be advantageous. Java-Annotations which have been introduced in Java 1.5 [18] are widely used for documentation and integration of plugins in the user interface. As mentioned in I-B the Java-platform enables writing plugins in any programming language with existing java-support (a full list can be found at [14]).

A. Plugin structure

Similar to ImageJ [19] OpenAnalyser plugins are java classes derived from a plugin superclass. The superclass-type determines the type of the plugin, e.g. whether it is used as a function for manipulating or analysing data, importing data or visualization of data. Algorithms are defined in the public call-function of plugin classes. The plugin system implements a mechanism similar to function overloading present in many programming languages. A comprehensive description of the plugin system can be found in the online manual [17]. Here, we briefly describe function overloading and parameter declaration.

1) Function-Overloading: The application of algorithms to data is the main focus for plugins in OpenAnalyser. A lot of algorithms are not limited to the application to a particular kind of data. A median filter for example could be applied to image, video or time series data. A plugin may define multiple call-functions, which can be distinguished by their parameter list. On invocation of the plugin the right call-function is selected by comparison with the data supplied by the user. For example, a plugin applicable to video data could be declared like this:

```
class ExamplePlugin extends GeneralFunction {
public void call (VideoDataset video) {} }
```

Of course algorithms are not restricted to work with only one dataset. Call-methods working with multiple datasets declare argument lists with more than one parameter.

2) Parameter declaration: Most algorithms need - beside the data - a number of input parameters, e.g. data like thresholds or external system parameters. Those parameters require integration into the user interface for setup. As mentioned above, parameters need to be stored as log entries so that they can be reviewed later. These problems are solved by a generic parameter editor which is able to read specially annotated Java-variables and generate a user interface. The following lines declare a floating point variable that can be used inside the algorithm and set by the user:

```
@ParameterDescription
public double a = 0;
```

A description of the parameter can be integrated into the user interface by using the dedicated field in the ParameterDescription-annotation.

IV. Applications

The following section describes typical applications of OpenAnalyser.

A. Time Series Analysis of ECG-data

A typical task in biosignal processing is ECG analysis and classification. A common first step is the detection of QRS-complexes (see Fig. 2) from which beat-to-beat-intervals

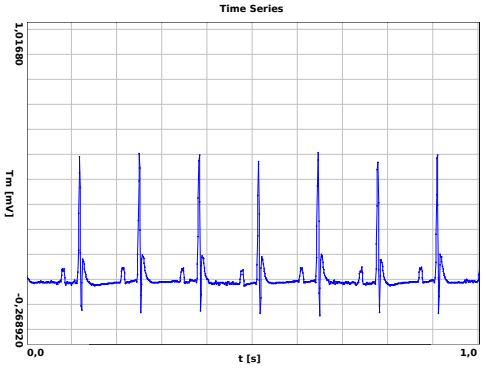

Fig. 2. An ECG-signal of a RyR2-R2474S Knock-in mouse

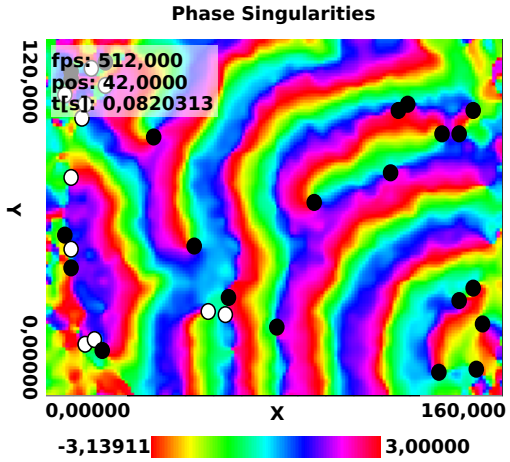

Fig. 3. Visualization of phase singularities in a cell culture experiment. Thanks to C. Richter for the data.

can be obtained. These beat-to-beat time series are then characterized by heart rate variability parameters [20] or other biomarkers [21]. The performance for detecting classifying pathological cases has to be assessed by means of statistical test, such as p-values, Wilcoxon-, T- and Mann-Whitney-U-tests. In future extensions of the time series analysis section of OpenAnalyser we shall include novel data analysis tools such as recurrence analysis, synchronisation detection and analysis of symbolic dynamics.

B. Phase Singularity Analysis

A widely used method to analyse excitable media is phase singularity analysis [22; 23]. There are several algorithms available to convert signals into phase data and detect phase singularities. For obtaining phase data from experimental video data OpenAnalyser provides the Hilbert transform which creates the analytic signal for a real time series. Phase singularity detection is done by searching for positions inside of the complex data where real and imaginary part have both zero crossings. Those positions are marked as singularities (see Fig. 3) and their topological charge is determined by computing the path integral over the gradient of the phase around a singularity.

For characterizing excitable media via phase singularity analysis, statistics over certain features of singularities like

liefetimes or mean distance from the point of their first appearance often is a useful tool. To determine these parameters it is necessary to combine phase singularity positions of adjacent frames to singularity trails. A simple approach for this problem is nearest neighbor detection which is implemented in OpenAnalyser. Afterwards several statistical analysis tools can be applied, including mean lifetime or mean velocity of singularities.

C. Excitable Media Dynamics

The Barkley model is a system of reaction-diffusion equations which can be used for the numerical simulation of excitable media. It is very similar to the FitzHugh-Nagumo model and was designed for fast numerical simulations of spiral waves [24; 25]. An important modification which allows wave breakup has been proposed by Bär and Eiswirth [26].

A plugin for creating data by simulating the Barkley model is provided within OpenAnalyser. It allows to model special conditions by varying model parameters a and b spatially. This can be done by converting image files into a suitable height-map.

V. Conclusion

OpenAnalyser combines the advantages of graphical user interfaces with a powerful and extensible framework for data analysis algorithms which may provide a useful and efficient tool for a large, interdisciplinary scientific community.

Acknowledgment

The authors would like to thank Amgad Squires and Daniel Hornung for many useful ideas. U.P. acknowledges financial support from the Max Planck Society (research contract *Analyse biophysikalischer Daten*). S.L. acknowledges support from the BMBF (FKZ 01EZ0905/6). The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007–2013 under grant agreement No. HEALTH-F2-2009-241526, EUTrigTreat (to S.L. and S.E.L.).

References

[1] "English wikipedia: List of numerical analysis software," accessed on 3/24/2010. [Online]. Available: http://en.wikipedia.org/wiki/List_of_numerical_analysis_software

[2] "Physionet," accessed on 3/24/2010. [Online]. Available: http://www.physionet.org/

[3] "Tstool," accessed 3/24/2010. [Online]. Available: http://www.dpi.physik.uni-goettingen.de/tstool/index.html

[4] "Ani2000," accessed 3/24/2010. [Online]. Available: http://cvp.physik.hu-berlin.de/index.html

[5] "Rqa analysis," accessed 3/24/2010. [Online]. Available: http://www.recurrence-plot.tk/

[6] "Gait cad," accessed 3/24/2010. [Online]. Available: http://sourceforge.net/projects/gait-cad/

[7] "Tocsy," accessed 3/24/2010. [Online]. Available: http://tocsy.agnld.uni-potsdam.de/

[8] "Biosignal tools," accessed 3/24/2010. [Online]. Available: http://biosig.sourceforge.net/

[9] "librasch," accessed 3/24/2010. [Online]. Available: http://www.librasch.org/

[10] "Imagej," accessed on 3/24/2010. [Online]. Available: http://rsbweb.nih.gov/ij/

[11] A. Schloegl, C. Vidaurre, E. Hofer, T. Wiener, C. Brunner, R. Scherer, and F. Chiarugi, "Biosig - standardization and quality control in biomedical signal processing using the biosig project," in *Proceedings of BIOSTEC*, 2008.

[12] J. Lewis and U. Neumann, "Performance of java versus c++," January 2003, accessed on 3/24/2010. [Online]. Available: http://www.idiom.com/ zilla/Computer/javaCbenchmark.html

[13] "Java as a scientific programming language," accessed on 3/30/2010. [Online]. Available: http://www.developer.com/java/other/article.php/631151/Java-as-a-Scientific-Programming-Language-Part-1.htm

[14] "Wikipedia: List of jvm languages," accesed on 3/30/2010. [Online]. Available: http://en.wikipedia.org/wiki/List_of_JVM_languages

[15] "The jython project," accessed on 3/30/2010. [Online]. Available: http://www.jython.org/

[16] "Jruby," accessed on 3/30/2010. [Online]. Available: http://jruby.org/

[17] A. Schlemmer, *OpenAnalyser*, Max-Planck-Institute for Dynamics and Self-Organization, 3 2010. [Online]. Available: http://www.bmp.ds.mpg.de/software.html

[18] B. Cornelius, "Java 5 catches up with c#," December 2005, accessed on 3/24/2010. [Online]. Available: http://www.barrycornelius.com/papers/java5/onefile/

[19] W. Burger and M. J. Burge, *Digital Image Processing*. Springer, 2008, iSBN: 978-1-84628-379-6.

[20] T. F. of The European Society of Cardiology, T. N. A. S. of Pacing, and Electrophysiology, "Heart rate variability," *European Heart Journal*, vol. 17, pp. 354–381, 1996.

[21] U. Parlitz, S. Berg, S. Luther, A. Schirdewan, and N. Wessel, "Classifying cardiac biosignals using order pattern statistics and symbolic dynamics," in *PROCEEDINGS OF THE 6TH ESGCO 2010*, 2010.

[22] P. A. e. a. Liu Y-B, "Spatiotemporal correlation between phase singularities and wavebreaks during ventricular fibrillation," *J Card. Elctrophysiol.*, vol. 14, pp. 1103–1109, 2003.

[23] C. R.H., Z. E.A., and P. A.V., "Phase singularities and filaments: Simplifying complexity in computational models of ventricular fibrillation," *P. Biophys. Mol. Bio.*, vol. 90, pp. 378–398, 2006.

[24] D. Barkley, "A model for fast computer simulation of waves in excitable media," *Physica*, vol. 49D, pp. 61–70, 1991.

[25] M. Dowle, R. Mantel, and D. Barkley, "Fast simulations of waves in three-dimensional excitable media," *Int. J. Bif. Chaos*, vol. 7, pp. 2529–2545, 1997.

[26] M. Bär and M. Eiswirth, "Turbulence due to spiral breakup in a continuous excitable medium," *Phys. Rev. E*, vol. 48, pp. R1635–R1637, 1993.