Journal of
open research software

Posch, S and Möller, B 2017 Alida – Advanced Library for Integrated Development of Data Analysis Applications. *Journal of Open Research Software,* 5: 7, DOI: https://doi.org/10.5334/jors.124

SOFTWARE METAPAPER

Alida – Advanced Library for Integrated Development of Data Analysis Applications

Stefan Posch and Birgit Möller
Institute of Computer Science, Faculty of the Natural Sciences III, Martin Luther University Halle-Wittenberg, Halle (Saale), DE
Corresponding author: Stefan Posch
(stefan.posch@informatik.uni-halle.de)

Data analysis procedures can often be modeled as a set of manipulation operations applied to input data and resulting in transformed intermediate and result data. The Java library `Alida` is providing an advanced development framework to support programmers in developing data analysis applications adhering to such a scheme. The main intention of `Alida` is to foster re-usability by offering well-defined, unified, modular APIs and execution procedures for operators, and to ease development by releasing developers from tedious tasks. `Alida` features automatic generation of handy graphical and command line user interfaces, a built-in graphical editor for workflow design, and an automatic documentation of analysis pipelines. `Alida` is available from its project webpage http://www.informatik.uni-halle.de/alida, on Github and via our Maven server.

Keywords: Data Analysis; Programming Framework; Implementation; Reusability; Graphical User Interfaces; Command Line Interface; Processing History; Java Software Library
Funding Statement: The development of `Alida` was supported by core funding from the Martin Luther University Halle-Wittenberg and the federal state of Saxony-Anhalt, respectively.

(1) Overview

Introduction

Automatic data analysis aims at cleaning, transforming, and modelling data to gain useful information in an application domain and for a specific problem statement. This process frequently requires the combination of various basic and advanced analysis steps into complex workflows, and several software tools for workflow design supporting this process on the user side are available [4, 11, 10, 1]. They for example target at distributed, grid and cluster computing, big data analytics, or on integrating data from different sources [2], and sometimes provide end-users with functionality for graphically combining analysis units into analysis workflows. However, solely applying and combining existing algorithms is not always sufficient to extract desired information from given data. Especially as progress in science and research is often linked to designing new experiments and acquiring new types of experimental data, sophisticated analysis requires the adaptation of existing or the development and investigation of new data analysis algorithms.

The development of such tools is usually performed by programmers in close collaboration with end-users from the application side, and a close interaction during the development process is essential. Consequently and independent of the application domain, the programmer is required to not only develop the algorithms themselves, but he is also enforced to provide handy user interfaces and integrate the user as close as possible into the development process, e.g., by frequently releasing software updates. Workflow tools like KNIME [1] and Triana [10] in principal support the extension of their functionality programmatically. However, since they mainly focus on the end-user programmers have to cope, e.g., with restrictions on available data types and complex APIs.

To overcome these drawbacks and in contrast to these tools, `Alida` (*Advanced Library for Integrated Development of Data Analysis Applications,* [7, 9]) is a Java library which specifically targets at programmers rather than end-users of data analysis tools. It seeks to optimally support programmers in the process of developing and releasing new data analysis algorithms in close collaboration with end-users. To this end `Alida` defines a framework which allows programmers to easily implement new data analysis functionality in a modular fashion. It defines an API based on a very general model of data analysis where manipulation and transformation of input data into intermediate and final result data is performed by *operators* with a certain functionality. Every operator is fully specified by a set of parameters subsuming input data and configuration settings

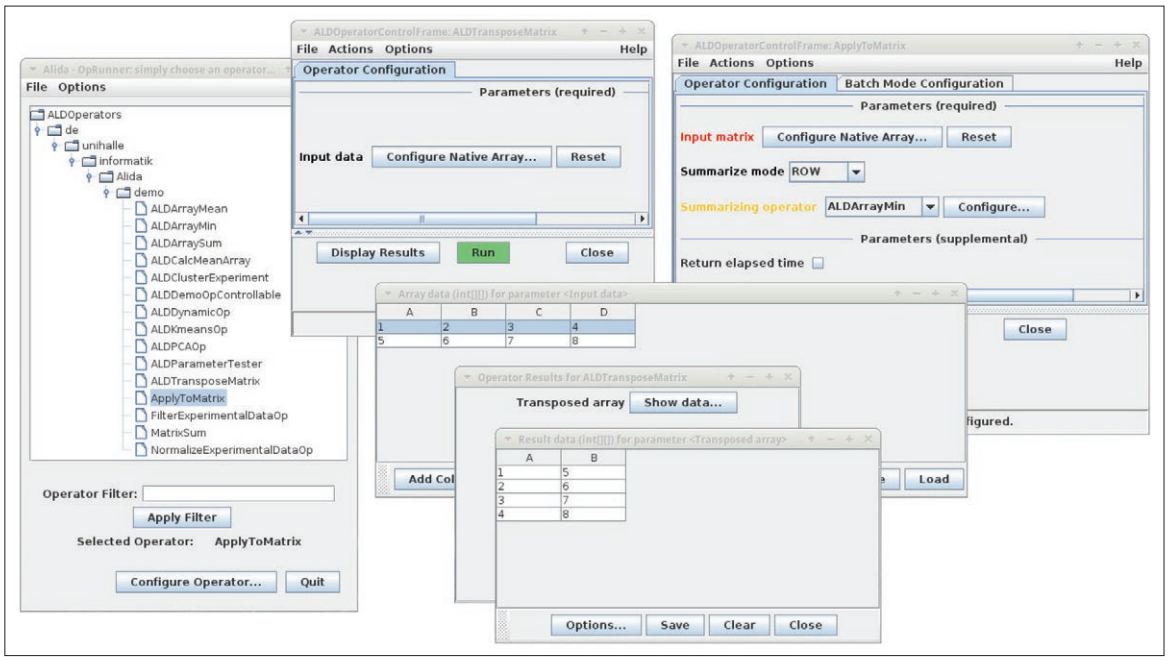

Figure 1: Screenshot of some automatically generated graphical user interfaces for configuration and execution of `Alida` operators.

for the functionality of the operator. During data analysis operators are applied sequentially, in parallel, or in a nested fashion to the input data and produce output data according to their configuration.

Based on this model `Alida` enforces only some few constraints on the implementation in order to release developers from reoccurring and tedious tasks like API design and user interface development. All operators share a common API for configuration and execution. On the one hand this facilitates reuse of operators on the code level and instant usage via the automatically generated command line user interface (see **Fig. 2**), e.g., for parameter optimization via a scripting language. On the other hand also graphical user interfaces are generated automatically (**Fig. 1**) fostering close end-user interaction and a tight feedback loop. Likewise all operators can automatically be included as potential building blocks in `Alida`'s built-in graphical workflow editor `Grappa` [3] (**Fig. 3**). Finally, since all operators are configured and executed by the same procedures automatic documentation of operator configurations and consequently also complete analysis pipelines is supported [6, 8].

The basic concepts of the `Alida` framework and its implementation in the Java library have proven their practical suitability and relevance as fundament of MiToBo, a toolbox of basic, intermediate and advanced image processing and analysis operators and applications [5]. All of the more than 150 operators in MiToBo are implemented as `Alida` operators taking full benefit of the unified interfaces and execution procedures and particularly of the automatically generated user interfaces.

Implementation and Architecture

The abstract class `ALDOperator` lays the foundation for `Alida`'s object-oriented design for data analysis. It is designed to enable `Alida`'s capabilities to automatically generate user interfaces, for graphical programming, and automatic documentation.

All operators to be implemented in `Alida` are required to extend this class. All data to be processed by an operator,

```
rigel:alida [161] java \
?    de.unihalle.informatik.Alida.tools.ALDOpRunner ApplyMatrix \
?    matrix='[[1,2,3],[4,5,6]]' \
?    summarizeOp='$ALDArrayMean:' summarizeMode=ROW \
?    summaries=-
summaries = [2.0,5.0]
rigel:alida [162]
```

Figure 2: Sample call of an operator from command line. The operator `ApplyToMatrix` is executed to apply an operator to a 2D array supplied on the command line. The parameter `summarizeOp` expects as value an `ALDOperator` which summarizes a 1D array, and here `ALDArrayMean` is specified computing the mean. The parameter `summarizeMode` is of enumeration type, and in this case row-wise summarization requested. The output is sent to standard output, but can be redirected to a file as well.

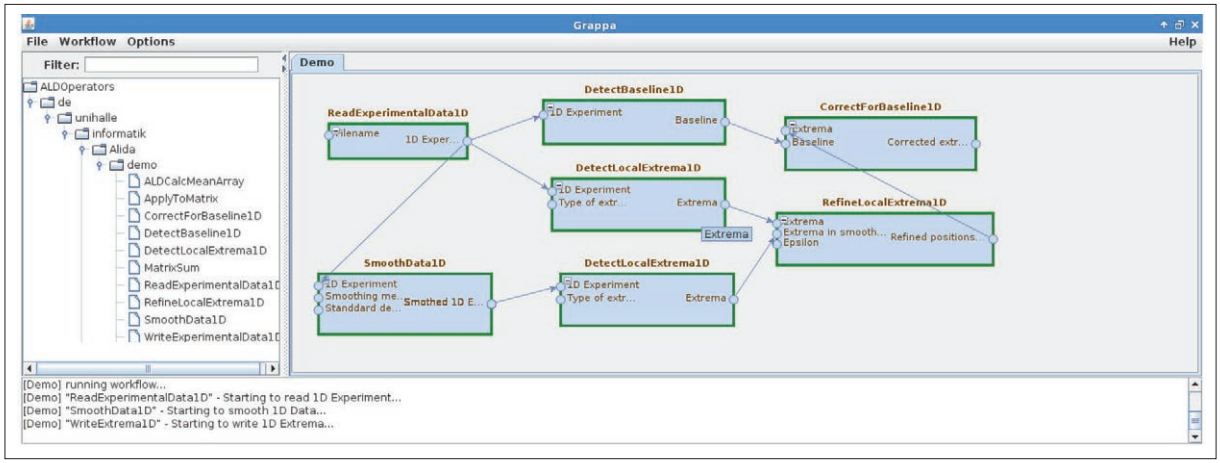

Figure 3: Screenshot of the graphical workflow editor `Grappa`, showing a demo workflow shipped with `Alida`. To the left a partial tree of available operators is displayed to choose from.

controlling its manipulation, or to be returned as result are consistently denoted as *parameters* in `Alida`. For each parameter a member variable is defined and Java's annotation mechanism is used to declare these members as parameters and specify their various properties. Java's reflection mechanism is exploited to implement methods for querying an operator for its parameters including data types and properties, as well as generic getter and setter methods for all parameters. The abstract method `operate()` of `ALDOperator` contains the data processing functionality and needs to be overridden by each operator implementation. The abstract class `ALDOperator` implements the method `runOp()` which is the only admissible way to invoke an operator. This allows to keep track of all operator invocations.

For all data processing algorithms implemented as `Alida` operators graphical and command line interfaces are instantly available to the users. To automatically generate these interfaces an operator needs to be queried for its parameters and their properties as stated above. In addition it is necessary to query values for parameters from the user, to instantiate parameter objects from these values, and to present output parameter values to the user, e.g., graphically or via console. As this depends on the specific data type and the set of potential parameter data types is unknown in advance, `Alida` incorporates a mechanism to link this I/O knowledge to specific data types. This is facilitated via so-called data I/O *providers* which provide the functionality for a given data type or set of data types and register to `Alida`'s framework using Java's annotations. Currently, `Alida` features general purpose providers for all primitive data types, enumeration types, arrays, collections, and so-called *parameterized classes*. An arbitrary class may be declared as parameterized class, and any subset of its member variables declared as *class parameters*, both via annotations. This is sufficient for `Alida`'s general purpose provider to handle this class as an operator parameter if providers for the class parameters exist.

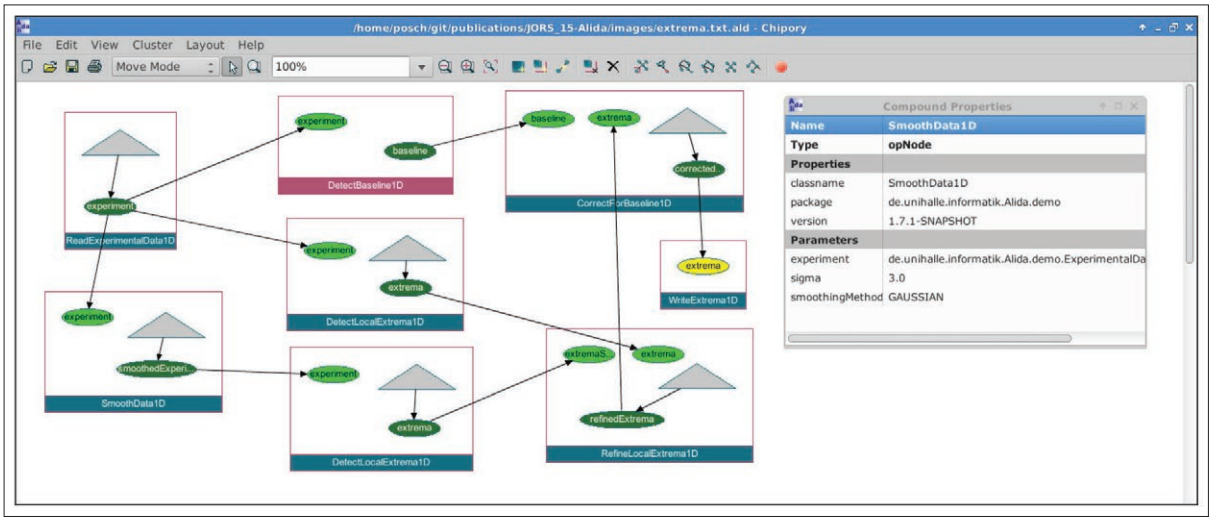

Figure 4: The processing graph for the workflow in Fig. 3. Each operator invocation is represented by a blue or red rectangle. A red rectangle indicates that an operator was *collapsed* to hide nested operator calls. Light and dark green ellipses are input and output ports respectively of an operator, gray triangles depict data ports representing newly generated data. To the right the information for the operator `SmooothData1D` is shown including the values of input parameters and software version.

Likewise operators may act as parameters of other operators. If necessary additional providers may easily be added without the necessity to modify `Alida`'s core. **Figs. 1** and **2** show examples for graphical respectively command line UIs automatically generated by `Alida`.

`Alida` extends the operator concept towards combining operators into more complex workflows. A workflow is defined as a combination of operators to be executed sequentially, in parallel, or in a nested fashion. This concept is implemented as the class `ALDWorkflow` which extends `ALDOperator`. The graphical programming editor `Grappa` is included in `Alida` to interactively design workflows in an intuitive fashion. The data processing pipeline is naturally modelled as a graph, where operators are represented by nodes, and the parameters of different operators are connected by edges to describe the flow of data. All data processing algorithms implemented as an `Alida` operator are right away available as operator nodes in `Grappa` and form the building blocks for workflows (see **Fig. 3** for an example). When connecting parameters of different nodes the validity is verified. For example, an input parameter may have at most one incoming edge, and the data types of parameters connected by an edge need to be compatible. Data propagated along an edge may be converted on user request if an appropriate converter is implemented. For example `Alida` includes functionality to convert an array to a collection. The set of converters may be extended in analogy to data I/O providers. In general the `operate()` method of a workflow object invokes all operators of the workflow in topological order and forwards output data between operators according to the data flow. In addition partial execution of the workflow is supported.

`Alida` also includes automatic process documentation of an analysis procedure which is supposed to contain all information necessary to recover the results from the same input data at a later point in time. Since each operator execution is realized invoking the generic `runOp()` method, the processing pipeline can be understood as a subgraph of the dynamic call graph of the analysis process. This call graph may also be interpreted as a hierarchical graph where each invocation of an operator is represented by a node. Besides the input data provided by the data flow between operators, in addition all control settings and also metadata like software versions are fully automatically retrieved during processing and represented. At any point in time the relevant portion of this processing graph may be retrieved and made explicit in terms of XML representations. This representation may be stored for archival purposes to, e.g., extract relevant information for publication. `Alida` also includes Chipory (see `Alida`'s homepage) to graphically display the processing graph and to inspect, e.g., parameter settings (see **Fig. 4** for an example).

Quality Control

The `Alida` library is actively developed since 2010 and has reached a mature state. The core has converged to a stable status and new features are integrated very diligently. The core functionality of `Alida` and particularly the components of the graphical user interfaces are mainly tested manually, partially relying on test operators specifically designed to test a certain functionality. Feedback may be submitted via a bug tracking system and using Github's pull requests. In addition, since `Alida` forms the base of the Microscope Image Analysis Toolbox MiToBo (http://www.informatik.uni-halle.de/mitobo), its development is also significantly triggered and supported by feedback, bug reports and feature wishes from the users of MiToBo [5]. This significantly adds to the robustness and stability of the `Alida` library. The tests and the use of MiToBo subsuming `Alida` have been performed on different operating systems (64-bit Linux, Windows XP and 7, OS X) and with different Java versions.

(2) Availability

Project Homepage

http://www.informatik.uni-halle.de/alida.

Operating System

Alida runs on different versions of Linux, OS X, and Windows.

Programming Language

Java, version 1.8.

Additional system requirements

None.

Dependencies

The `Alida` distribution is shipped with all libraries required to make use of `Alida`'s complete functionality. For own developments based on `Alida` a Maven server[1] hosts the latest artifacts keeping track of dependencies automatically.

List of contributors

Birgit Möller
Stefan Posch

Software location

Archive

Name: Zenodo Research Archive
Title: Alida – Advanced Library for Integrated Development of Data Analysis Applications: v2.7
URL: https://zenodo.org/record/47586
Persistent identifier: https://doi.org/10.5281/zenodo.47586
Licence: GNU General Public License, Version 3
Publisher: Stefan Posch, Birgit Möller
Artifact Version: 2.7
Date published: 15/03/2016

Code repository

Name: Github
Title: alida
Identifier: https://github.com/alida-hub/alida
Licence: GNU General Public License, Version 3
Publisher: Birgit Möller, Stefan Posch
Date published: 14/06/2015

Maven repository server

Name: Apache Archiva Repository Server
Title: de.unihalle.informatik.Alida
Identifier: https://moon.informatik.uni-halle.de/archiva/
Licence: GNU General Public License, Version 3
Publisher: Birgit Möller, Stefan Posch
Date published: snapshots and releases are continuously published

Language

English

(3) Reuse Potential

The overall target of the `Alida` library is to provide a framework for developing modular, easy-to-use and particularly reusable data analysis software. Consequently, re-usability is the inherent key paradigm which coins the design and implementation of `Alida` in all respects. Moreover, this re-usability is not restricted to a specific research field, rather `Alida` is suitable for developments in all domains where data analysis coincides with `Alida`'s concept of operators performing data manipulations.

Besides this conceptual perspective `Alida` also aims to foster re-usability from a technical point-of-view. As mentioned above Maven is used to automatically resolve dependencies. Moreover on the website of `Alida` in the 'Downloads' section a template Maven project[2] can be found which is readily configured for immediate use in own projects. Finally, the open source strategy of `Alida` and the GPL licensing concept inherently guarantee a high degree of flexibility and adaptivity of `Alida` which renders it easy to even adjust the core functionality to new areas and fields of application if necessary.

Acknowledgements

The authors would like to thank Oliver Greß, Markus Glaß, and Danny Misiak for numerous valuable discussions on concepts, architectures and features of `Alida` over the whole time of the library's formation, implementation and fine-tuning.

Notes

[1] https://moon.informatik.uni-halle.de/archiva/#welcome.

[2] http://www2.informatik.uni-halle.de/agprbio/alida/downloads/alida-project-template-maven-1.2-src.zip.

Competing Interests

The authors have no competing interests to declare.

References

1. **Berthold, M R, Cebron, N, Dill, F, Gabriel, T R, Kötter, T, Meinl, T, Ohl, P, Thiel, K** and **Wiswedel, B** 2009 KNIME – The Konstanz Information Miner: version 2.0 and beyond. *ACM SIGKDD Explorations Newsletter,* 11(1): 26–31, DOI: https://doi.org/10.1145/1656274.1656280
2. **Curcin, V** and **Ghanem, M** 2008 Scientific workflow systems – can one size fit all? In *2008 Cairo International Biomedical Engineering Conference,* pages 1–9, Dec 2008. DOI: https://doi.org/10.1109/cibec.2008.4786077
3. **Kirchner, S, Posch, S** and **Möller, B** 2012 Graphical programming in Alida and ImageJ 2.0 with Grappa. In *Proc. of ImageJ User & Developer Conference,* pages 138–143, Mondorf-les-Bains, Luxembourg, October.
4. **Ludäscher, B, Altintas, I, Berkley, C, Higgins, D, Jaeger, E, Jones, M, Lee, E A, Tao, J** and **Zhao, Y** 2006 Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience,* 18(10): 1039–1065,. DOI: https://doi.org/10.1002/cpe.994
5. **Möller, B, Glaß, M, Misiak, D** and **Posch, S** 2016 MiToBo – A Toolbox for Image Processing and Analysis. *Journal of Open Research Software,* 4(1): p. e17, DOI: https://doi.org/10.5334/jors.103
6. **Möller, B, Greß, O** and **Posch, S** 2011 Knowing What Happened – Automatic Documentation of Image Analysis Processes. In Crowley, J, Draper, B and Thonnat, M. (Eds.), *Proceedings of 8th International Conference on Computer Vision Systems,* volume 6962 of LNCS, pages 1–10, Sophia Antipolis, France, Springer.
7. **Möller, B** and **Posch, S** 2013 A Framework Unifying the Development of Image Analysis Algorithms and Associated User Interfaces. In *Proc. of 13th IAPR International Conference on Machine Vision Applications,* pages 447–450, Kyoto, Japan.
8. **Posch, S** and **Möller, B** 2012 Automatic Generation of Processing Histories using Alida. *In Proc. of ImageJ User & Developer Conference,* pages 218–221, Mondorf-les-Bains, Luxembourg, October.
9. **Posch, S** and **Möller, B** 2016 Design and Implementation of the Alida Framework to Ease the Development of Image Analysis Algorithms. *Pattern Recognition and Image Analysis,* 26(1): 181–189, DOI: https://doi.org/10.1134/S105466181601020X
10. **Taylor, I, Shields, M, Wang, I** and **Harrison, A** 2007 *The Triana Workflow Environment: Architecture and Applications,* pages 320–339. Springer, London.
11. **Wolstencroft, K, Haines, R** and **Fellows, D** et al 2013 The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research,*. DOI: https://doi.org/10.1093/nar/gkt328

How to cite this article: Posch, S and Möller, B 2017 Alida – Advanced Library for Integrated Development of Data Analysis Applications. *Journal of Open Research Software,* 5: 7, DOI: https://doi.org/10.5334/jors.124

Submitted: 15 March 2016 **Accepted:** 13 February 2017 **Published:** 23 March 2017

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS