A Framework Unifying the Development of Image Analysis Algorithms and Associated User Interfaces

Birgit Möller and Stefan Posch
Institute of Computer Science, Martin-Luther University Halle-Wittenberg,
Von-Seckendorff-Platz 1, 06099 Halle (Saale), Germany
{birgit.moeller,stefan.posch}@informatik.uni-halle.de

Abstract

Solving image analysis problems not only requires the development of suitable sets of algorithms to produce desired result data, but also demands for suitable user interfaces (UIs) to foster use in practice. Here we present our library Alida *which aims to promote the deployment of UIs by featuring their automatic generation from algorithm code.* Alida *supports command line and graphical UIs (GUIs), and ships with a graphical editor for designing more complex workflows. Enforcing only a small set of rules to obey* Alida *significantly reduces implementation overhead for developers and allows for focusing on algorithm rather than UI design. The suitability of* Alida*'s concept for real-life applications is shown by the library* MiToBo *for biomedical image analysis implemented based on* Alida.

1 Introduction

The process of automatically analyzing image data can generally be understood as a sequence of individual analysis steps applied to data. These steps are performed sequentially, in parallel, or in a nested fashion and transform given input data into application-specific result data. Often each single step of such analysis pipelines is associated with a specific processing unit or *operator* implementing functionality and performing the actual work on the data.

Consequently, the development of complex image analysis pipelines comprises two issues. On the one hand it subsumes the development of suitable operators, and on the other hand it requires their combination into pipelines through which the data is propagated to solve a task at hand. Besides these two fundamental issues the availability of suitable UIs on the programming as well as on the user level is equally important. Particularly the latter ones are inevitable to pave the path for newly developed algorithms into practice where non-expert users are highly interested in using the software to solve their problems.

Although a great consent is to be expected within the community regarding the necessity of handy user interfaces to be available, nevertheless their development is too often neglected in practice. The main reason for this is the significant amount of additional workload for interface design and implementation.

In this paper we present an approach for closing the gap between algorithm and pipeline design on the one side, and the development of UIs on the other. Our key contribution is to provide a development environment in terms of a Java library which inherently integrates the development of algorithms and UIs, and which is publicly available under GPL license[1]. This library named *Alida* defines operators as key components and requests developers to implement these by following general rules. Each operator has to support inquiries for its input and output parameter data types and values. In addition operators have to be invoked by a unified procedure enabling generic handling of all operators.

These guidelines may seem to restrict developers in their design options, but they only enforce a minimal set of rules as will become obvious in Sec. 3. Moreover, by embedding new operators into the environment provided by *Alida* all operators are natively accessible through a unified API on the programming level, rendering their use within code quite easy. Finally, these rules lay the foundation for automatically generating graphical as well as command line UIs for all operators in a generic fashion (Sec. 4). They also enable the export of operators into more complex applications like graphical editors for pipelines in a straightforward manner. Accordingly, such an editor is also included in *Alida* and consequently extends the comfortable development and application of single operators towards user-friendly graphical pipeline design (Sec. 5).

2 Related Work

For many image analysis problems well-established algorithmic solutions have emerged over time. They are often collected in libraries like OpenCV [3] or ITK [6], or in commercial tools like Matlab. While this fosters wide-spread use, the lack of handy user interfaces still forms a barrier. Although OpenCV and Matlab provide basic components like windows and buttons for interaction, the implementation of GUIs is still left to the developer of a certain application.

For the development in Java libraries and toolkits like JAI [11] and ImageJ [1], which is a toolbox of image analysis algorithms widely-used withing the biomedical community, aim to simplify operator development providing an embedding framework. While JAI is focused on unifying parameter handling and operator invocation, the new version 2.0 of ImageJ[2] also includes mechanisms for automatic generation of GUIs. Our concept *Alida* surpasses this by featuring advanced concepts for handling operator parameters (cf. Sec. 4), handy command-line tools, and support for automatic process documentation [9].

For designing analysis pipelines various tools are available. While the graphical workflow editor Kepler[3] does not target at a specific field of application,

[1]http://www.informatik.uni-halle.de/alida/
[2]http://developer.imagej.net/
[3]https://kepler-project.org/

KNIME [2] is dedicated to data mining and natively supports image processing. However, extending these tools with new functionality is not always straightforward. Our library *Alida* is designed for data analysis in general, but we have also deployed the image processing library *MiToBo* based upon *Alida* and dedicated to biomedical imaging which proves the suitability of our concept in practice (Sec. 6). It is deeply integrated into ImageJ overcoming the need for non-expert users to adapt to a completely new environment for applying operators and developing their workflows.

3 Basic Concepts

As outlined in the introduction, *Alida* defines *operators* as the basic units for data analysis. Taking an object-oriented approach an operator is expected to extend the common super class `ALDOperator` and to implement its functionality by overriding specific methods.

All data to be processed, controlling manipulation, or to be returned as result from an operator are consistently denoted as *parameters*. The role of a parameter is specified via its *direction*, which may be `IN`, `OUT`, or `INOUT`. A typical example is a filter applied to an input image (direction `IN`) where the filtered image is returned in a newly allocated data structure as a parameter with direction `OUT`. If the filter acts destructively in place, this is described by a single parameter of direction `INOUT`. A parameter controlling the filter operation, e.g. a bandwidth, is provided as an `IN` parameter. Parameters of direction `IN` and `INOUT` may be either *required* or *optional* to simplify operator configuration. In addition *Alida* supports *supplemental* parameters which in contrast to required and optional ones must not influence processing results. Examples include flags to control debug messages or intermediate results to be returned from operators after execution.

For defining and accessing the parameters of an operator *Alida* makes use of Java's annotation mechanism. Parameters can easily be added by simply annotating member variables of the operator class. In addition, the annotations also allow to easily query an operator for all its parameters including their type, role and current value at run time using methods supplied by the common super class `ALDOperator`.

To make use of an operator's functionality an instance of the class needs to be created and its `IN` and `INOUT` parameters have to be set. Processing is invoked calling the generic `runOp()` method supplied by `ALDOperator`. This method subsumes an automatic parameter validation, e.g. it checks whether all required parameters are provided, and optionally enforces operator-specific constraints, e.g. admissible ranges of parameter values. Subsequently the operator's `operate()` method is called performing data manipulations, and finally the results are made available via the output parameters of the operator.

As outlined in the introduction image analysis problems usually require a combination of operators to be applied to data rather than only a single one. Consequently *Alida* natively supports the development of complete pipelines for data processing by providing the class `ALDWorkflow`. This class represents a processing pipeline as a set of operators. It allows for establishing links between `OUT` and `IN` parameters of different operators by which a flow of data and control and thus a pipeline can be realized.

A workflow in *Alida* also features entry and exit points for data into and out of the whole pipeline. These points have essentially the same role as parameters for operators, hence, they share exactly the same properties, e.g., have a direction and may be required or optional. This naturally implies to implement `ALDWorkflow` extending the super class `ALDOperator` of all operators. The `operate()` method of a workflow object invokes all included operators in topological order and forwards output data between operators according to the data flow. To facilitate graphical programming, the class `ALDWorkflow` also supplies methods to invoke only part of the processing pipeline, to save and load workflows, and offers an event mechanism for GUI components to register, thus facilitating a Model-View-Controller architecture (cf. Sec. 5).

4 Automatic User Interface Generation

The operator concept of *Alida* with its clearly defined specification of parameters and its standardized invocation procedure lays the basis for generic implementation of UIs. To facilitate generic configuration and execution of operators such UIs are required to allow for the input of parameter values, to invoke the operator, and finally to publish the results.

In *Alida* the *Model-View-Control* design pattern [4] is used to achieve maximal independence between the operators implementing the functionality, the I/O of data objects, and the graphical and textual UIs. As input and output mechanisms of individual data items are data type specific, I/O functionality is also completely encapsulated in data I/O *providers* hiding any data type specific knowledge from the generic viewers.

To endorse the development of new functionality *Alida* already includes various providers which facilitates I/O for a wide variety of Java objects out of the box and overcomes the need for programmers to implement I/O capabilities. Besides providers for all primitive data types and arrays *Alida* subsumes general purpose providers for all enumeration types, collections, and so-called *parametrized classes*. An arbitrary class may be declared as parametrized class, and any subset of its member variables as *class parameters*, both via annotations. This is sufficient for *Alida*'s general purpose providers to handle this class as an operator parameter. Likewise operator objects by itself may act as parameters of other operators. Only specialized classes like images or contour sets require additional providers to be implemented, but these can easily be added to the library with no need to modify the code of *Alida*.

4.1 Command line

Building on this infrastructure *Alida* features a command line operator runner (CLR) to invoke all operators via console or scripts. All input parameters are supplied as arguments by 'name=value' pairs. To ease the handling of class inheritance, parametrized classes and operators as parameters in a generic fashion, the CLR features a flexible parser for argument preprocessing. It allows for parsing CLR calls like the one shown in Fig. 1. The operator `SnakeOptimizerCoupled`

```
java OpRunner SnakeOptimizerCoupled initialSnakes=RoiSet.xml inImg=cell.tif
 outSnakes=snakesOut.xml snakeOptimizer='$SnakeOptimizerSingleVarCalc:{energySet=
 {energies=[$MTBSnakeEnergyCD_CVRegionFit:{lambda_in=1.0,lambda_out=5.0}],weights=[1.0]}}'
```

Figure 1. Example call of an operator from command line. The operator `SnakeOptimizerCoupled` called here among others takes an operator of type `SnakeOptimizerSingleVarCalc` as input parameter.

for multiple snake segmentation not only takes initial snakes and an image as input, but also an operator instance of `SnakeOptimizerSingleVarCalc` dealing with a single snake (cf. Fig. 2, right). The syntax of the individual value strings is defined by the specific I/O providers they are finally passed to and which, by convention, also allow values to be read from file. Analogously output parameters are specified as name–value pairs allowing to, e.g., redirect output into files, as an alternative to formatting the values onto standard out.

4.2 Graphical user interface

A GUI is supposed to support graphical configuration and execution of operators. Unlike the command line UI a graphical front-end to choose, configure and execute operators is also well-suited for interactive exploration of image processing, e.g. for online inspection of the effects of parameter changes during rapid-prototyping. The GUIs currently provided by *Alida* are implemented based on Swing, but this is not a fundamental restriction as other frameworks likewise web interfaces can be added in a straightforward way.

As soon as the user has selected an operator from the choice of available operators, a window to configure it and control its execution is automatically generated (cf. Fig. 2 on the right, which shows the GUI for the operator `SnakeOptimizerSingleVarCalc` introduced above). To this end the operator is first queried for the types of its input and output parameters. Subsequently for each parameter a corresponding graphical component is generated using the provider mechanism outlined above. All components are arranged in a frame whereas required, optional and supplemental parameters are grouped into different sections. Besides these components for operator configuration allowing for user inputs, the window also contains buttons for controlling operator execution and enabling interaction (if the operator supports that). In addition a menu-bar is available containing items, e.g., for accessing online help or saving and loading the configuration to and from file. After execution of an operator result data is displayed to the user again adopting the provider mechanism for generating graphical components for each output parameter.

The configuration and control window acts as controller and likewise observer of the underlying operator and its status. The configuration status of the operator is synchronized online with the window and vice versa. E.g., parameters that are required, but do not yet have suitable values are marked in red, and the color of the run button indicates whether the operator is ready for execution or not. On the other hand changes in the parameters made by the user are directly propagated to the operator and induce an instant update of its configuration and potentially also its status.

5 Graphical Programming

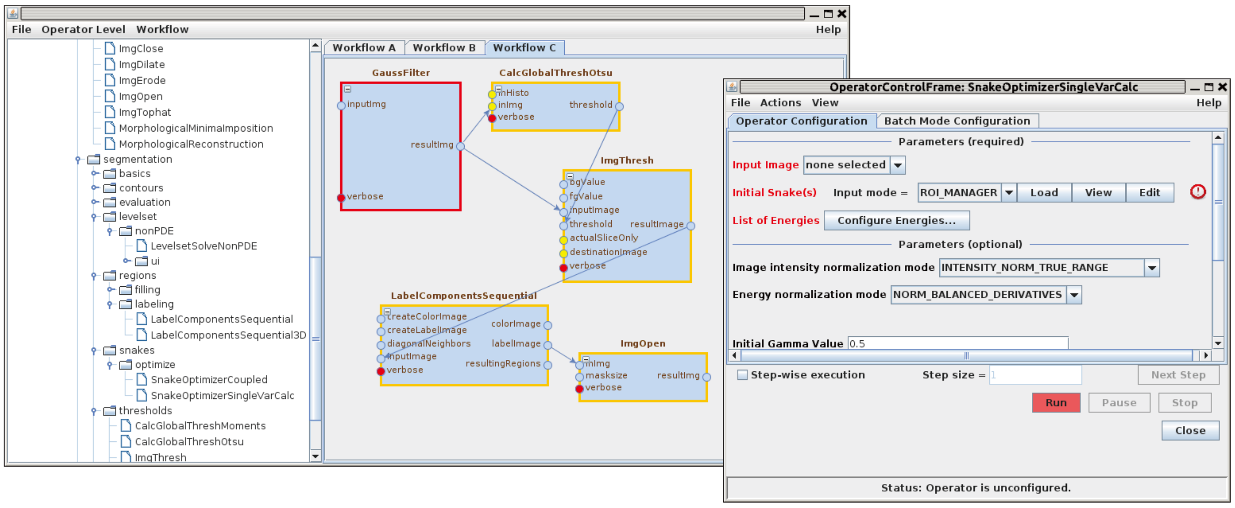
Designing more complex analysis pipelines featuring the combination of various operators can be facilitated in a comfortable, intuitive and user-friendly way by graphical programming editors. The underlying idea of these tools is to translate the design process into a graph editing task. Operators are represented by nodes, and edges in-between indicate the flow of data and control. Obviously *Alida* is an optimal foundation for such an editor. Its workflow concept is natively qualified as basis since it inherently defines a computational model of workflows. And also the clearly defined concept of operators renders it very easy to adopt operators as basis for configurable nodes within a graph data structure associated with a processing pipeline. They are handled in a generic fashion similar as in the context of UI generation discussed earlier.

The graphical editor *Grappa* included in *Alida* is built on these ideas (Fig. 2). It automatically includes all available operators into a menu (left of Fig. 2) from where the user can select operators for a workflow. For each operator a corresponding node is generated on the workbench showing the parameters of the operator as ports. These ports can be linked by edges resulting in a working cycle intuitive also for non-experts. For graphical node configuration each node is linked to a configuration window where the same components and mechanisms as for automatic GUI generation are used. Once pipeline design and configuration are completed the workflow graph can be executed either completely or in parts, e.g., only up to a certain node. After termination the results are displayed to the user again reusing mechanisms from GUI generation. Similar to the concepts of graphical operator configuration and control, the editor acts as controller and observer. The node color is updated according to the current status of the underlying operator in synchrony with changes in the configuration window which yields an intuitive visual guidance in pipeline configuration and execution.

6 Alida in Practice

The development of image analysis algorithms for biomedical applications natively requires deep links between algorithm developers and (non computer scientist) users. Algorithmic improvements and adaptations can most of the time best be done based on direct practical evaluations in the lab, and the frequent introduction of new imaging techniques and types of data both request for topical availability of appropriate software.

We meet these requirements by implementing our image analysis algorithms based on *Alida*, i.e. each algorithm is implemented as operator. The set of all operators is collected in the toolbox *MiToBo* which inherently supports execution of all available operators from command line as well as graphically. For straightforward usage the graphical front-end to select,



Input Image none selected

Initial Snake(s) Input mode = ROI_MANAGER

List of Energies

Image intensity normalization mode INTENSITY_NORM_TRUE_RANGE

Energy normalization mode NORM_BALANCED_DERIVATIVES

Initial Gamma Value 0.5

Step-wise execution Step size = 1

Status: Operator is unconfigured.

Figure 2. Screen-shot of the graphical editor for user-friendly pipeline design with its selection menu and workbench (left), and an automatically generated configuration and control window for an operator (right).

configure and execute operators is available as plugin[4] for ImageJ, a tool extensively used by our biomedical cooperation partners. This way algorithmic improvements and extensions can directly be released for practice in terms of new ImageJ plugins. By this users get software for current problems soon, and the developers earn topical feedback, in conclusion yielding great benefits for both sides.

Over the years a broad collection of algorithms for basic image processing tasks like morphological operations, filtering or labeling, as well as for specific areas of application, e.g., for sub-cellular particle detection [8], cell segmentation based on active contours [10, 7], or analysis of scratch assay images [5] have been developed. In addition the toolbox subsumes a plugin version of the editor *Grappa* (Sec. 5). Besides, non-expert users also benefit from some user-friendly built-in features of *Alida* not discussed here in detail, e.g., the automatic documentation of parameters used in analysis procedures simplifying communication between developers and users [9], or the option to filter as well the set of operators in the graphical selection menu as also the parameters of an operator displayed in the GUI.

7 Conclusions

Alida offers an integrated approach for unifying the development of image analysis algorithms and more complex analysis pipelines on the one hand and intuitive user interfaces on the other. Its underlying concept of operators lays the foundation for easy use of operators through a unified API on the programming level as well as for automatic generation of command line and graphical user interfaces on the user level. The concept results in a great flexibility which also becomes obvious in the graphical editor *Grappa* included in *Alida*. For its implementation *Alida*'s functionality with regard to operator execution and pipeline design had to be made available through appropriate graphical components, however, did not induce any conceptual changes in the underlying core. And the flexibility is not yet exhausted. It is obvious that the workflows of *Alida* being by themselves operators already inherently support the concept of hierarchical workflows which is currently being transferred to and implemented in the graphical editor. Finally, a batch mode is currently under development for automatically running operators on sets of values for specified input parameters to further ease algorithm tuning for developers and algorithm usage for non-expert users.

[4]http://www.informatik.uni-halle.de/mitobo/

References

[1] Abramoff, M.D., Magelhaes, P.J., Ram, S.J.: Image processing with ImageJ. Biophotonics Int 11(7), 36–42 (2004)

[2] Berthold, M.R., et al.: KNIME - the Konstanz Information Miner: version 2.0 and beyond. SIGKDD Explor. Newsl. 11(1), 26–31 (Nov 2009)

[3] Bradski, A.: Learning OpenCV: Computer Vision with the OpenCV Library. O‘Reilly Media (2008)

[4] Fowler, M.: Patterns of Enterprise Application Architecture. The Addison-Wesley Signature Series (2003)

[5] Glaß, M., et al.: Cell migration analysis: Segmenting scratch assay images with level sets and support vector machines. Pattern Recognition 45(9), 3154–3165 (2012)

[6] Ibanez, L., Schroeder, W., Ng, L., Cates, J.: The ITK Software Guide, 2. edn. (November 2005)

[7] Möller, B., Posch, S.: Comparing active contours for the segmentation of biomedical images. In: IEEE Int. Symp. on Biomedical Imaging. pp. 736–739 (2012)

[8] Möller, B., et al.: Adaptive segmentation of particles and cells for fluorescent microscope imaging. In: VISIGRAPP 2010, Revised Selected Papers of Int. Joint Conf. on Comp. Vision, Imaging and Comp. Graphics. Theory and Appl. vol. 229, pp. 154–169 (2011)

[9] Möller, B., Greß, O., Posch, S.: Knowing what happened - automatic documentation of image analysis processes. In: Proc. of Int. Conf. on Comp. Vision Systems. LNCS, vol. 6962, pp. 1–10. Springer (2011)

[10] Möller, B., Stöhr, N., Hüttelmaier, S., Posch, S.: Cascaded segmentation of grained cell tissue with active contour models. In: Proc. of Int. Conf. on Pattern Recognition (ICPR). pp. 1481–1484 (2010)

[11] Sun Microsystems, Palo Alto, CA 94303, USA: Programming in Java Advanced Imaging (1999), Rel. 1.0.1