Stefan Posch: Automatic Generation of Processing Histories using Alida

Abstract

Automatic analysis of biomedical images typically involves the application of several processing steps, sequentially as well as in parallel, to produce the final interpretation of the data. Besides these results, e.g. images or tables, it is also important to document the processing pipeline for later verification, reconstruction, or publication. Such a documentation subsumes not only which processing steps were applied in which sequence to which data, but also parameter settings and software versions of the individual steps.

If each processing step is realized by a function call, the processing pipeline can be understood as a subgraph of the dynamic call graph of the analysis process. This DAG may also be interpreted as a hierarchical graph, where each processing step contains nested invocations of child processing steps as inner nodes.

The Java library Alida (Advanced Library for Integrated Development of Data Analysis Applications) supports

fully automatic generation of processing histories. These processing histories include all manipulations performed, involved input and output parameters, the flow of data, and also the software version applied in each processing step. This processing history can be stored as a processing graph persistently as an XML file accompanying the resulting output file, e.g. an image or table. The XML representation is based on graphML and extended to satisfy Alida's needs. The graph representation of the history can for example be exploited visually. We extended Chisio, a free editing and layout tool for hierarchical graphs, for history graphs yielding Chipory which is delivered with Alida.

To implement this concept all processing steps are implemented as classes extending the abstract class ALDOperator. Data to be processed, controlling data manipulation, or to be returned as result

are denoted as parameters. In addition, Alida supports supplemental parameters which by definition must not influence the processing results. To implement an operator it is only necessary to define its parameters with Java's annotation mechanism using an extended version of the @Parameter annotation of ImageJ 2.0, and to supply its functionality implementing the abstract method operate(). To actually invoke an operator, an instance of the operator class is created, the input parameters of this object are set, and subsequently the method runOp() supplied by ALDOperator is called. Besides invoking the operate() method, the runOp() method creates for each invocation of an operator a node in the implicit processing graph and stores all non-supplemental parameter objects in a weak hash map for later reference. Furthermore, input parameters originating from a previous operator invocation are linked to the corresponding node to reflect the data flow within the processing pipeline. To this end, for each data object the originating operator is stored in a weak hash map. This map is updated as data objects are passed to or from further operator invocations.

Besides automatic documentation Alida provides automatically generated graphical user and

command line interfaces for operators. As the concept of a data processing pipeline is very general Alida is

applicable to all domains of data processing. Alida is freely available under the GPL V3 at

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

Keywords

Process Documentation, Processing Graph, XML, Alida, ImageJ

Short CV

Stefan Posch received his Diploma and Doctoral degrees in Computer Science from the University of Erlangen-Nürnberg, Germany, in 1985 and 1989. From 1990 to 1991, he held a postdoctoral position at the International Computer Science Institute, Berkeley, California. In 1991, he joined the Applied Computer Science Group at the University of Bielefeld, Germany. Since 1999, he is Professor at the Institute of Computer Science at the Martin Luther University Halle-Wittenberg. Research interests include computer vision, pattern recognition and bioinformatics.

Administrative data

Presenting author: Stefan Posch
Organisation: Institute of Computer Science, Martin Luther University Halle-Wittenberg

co-authors: Birgit Möller