Get some Coffee for free – Writing Operators with RapidMiner Beans

Christian Bockermann and Hendrik Blom
Technical University of Dortmund
Artificial Intelligence Group
{christian.bockermann,hendrik.blom}@udo.edu

Abstract

RapidMiner has become a valuable tool for business intelligence as well as in highschool education. Its plugin mechanism allows for creating powerful extensions and various plugins exist for different applications, data formats and learning schemes.

In this paper we present the RapidMiner Beans library, a simple library and plugin that aims at simplifying the development and documentation of custom operators using standard Java technologies like annotations and JavaBeans conventions.

One of the main objectives is to allow for the developer to solely focus on the operator development without caring about the house keeping required. For demonstration we give an example implementation that shows the benefits of the RapidMiner Beans.

1 Introduction

RapidMiner is a powerful tool for various data analysis and business intelligence tasks. It provides a wide range of data mining and machine learning algorithms, data pre-processing operators and evaluation methods. One of its strongest qualities is the plugin architecture, which allows for extending RapidMiner with additional functions using RapidMiner *Extensions* or *Plugins*.

These plugins add new operators and functions to the core RapidMiner suite and are implemented within the RapidMiner programming API. Over the years this API has evolved to a powerful – yet sometimes complicated to learn – instrument. Besides their core implementation, operators need to be registered within the operator descriptions file and various additional meta data files need to be defined for creating a custom plugin.

When developing extensions or plugins for RapidMiner, we often see students and new developers struggling with the same hurdles, a very big one of which is related to the plugin definition files. A second unpleasant burden is the way to document operators.

Whereas the former issue can be solved by some investment in setup time of the development environment, the latter causes the more severe follow-up as it discourages developers/students to properly document their achievements. This effectively feeds another hurdle – usage of the new operators within the student/RapidMiner community.

With the RapidMiner Beans plugin/library we approach these issues by

(a) lowering the effort required to write and deploy a new operator

(b) providing a simple tool for documenting that operator.

One of our superior goals was to achieve these objectives by working as close to the existing path of developing RapidMiner operators as possible – just with a little simplification.

In addition to this, we seek to feed some coding guidelines by following well-accepted conventions such as JavaBeans[1] and the use of Java's annotations mechanism. This allows for creating new operators by creating a single Java class file as well as additionally creating a single text file for documentation. Everything else will be taken care of by the RapidMiner Beans framework.

The rest of this paper is organized as follows: Section 2 gives a short high-level overview of the RapidMiner Beans library and the JavaBean conventions used within. Following that we will give a walk-through example for implementing and loading a RapidMiner operator simply with annotations in Section 3. Next we will demonstrate how to document our operator using the *Markdown* wiki-like dialect. Finally we will give some outlook on future work.

2 RapidMiner Beans – Overview

The RapidMiner plugin meachnism requires operators to be defined within a descriptive XML file. This file is referenced in RapidMiner extension files and used to register operators of a plugin at the start of RapidMiner. For each operator, the XML file contains

- the operator's name,
- a short description of the operator,
- the class implemeting the operator, and
- the group in which the operator is displayed within RapidMiner.

The RapidMiner Beans library facilitates two aspects: It is a standalone library that provides a small set of Java annotations. This allows to define all of the operator properties directly within the class by using class-level annotations.

As a second aspect the RapidMiner Beans library implements a plugin itself. At RapidMiner start-up time this plugin will check for classes annotated with the aforementioned annotations and register these in RapidMiner. This allows for adding new operators by simply creating annotated Java classes.

2.1 JavaBeans and Parameter Annotations

Partially following the JavaBeans conventions, operators can be equipped with `get`- and `set`-methods. By annotating these methods with field- or method-level `@ParameterInfo` annotations, this allows for automatically extracting RapidMiner parameter type lists.

With the `@ParameterInfo` annotation all basic properties of parameters that are required for displaying parameters within the RapidMiner user interface can be defined. This further decouples the operator implementation from the GUI layer aiming towards a cleaner separation of view and logic.

2.2 Operator Documentation

Finally, the documentation of operators is another important aspect of the RapidMiner Beans library. For this, we focus on the simple *Markdown* format, a simple wiki-like markup language that has recently become popular in portals such as `github.com`, and which easily translates to HTML.

For documentation, the RapidMiner Beans library requires a simple text file to be created for each operator, with the same name as the operators Java file but with the `.md`-extension instead.

This simple convention requires two files to be touched for creating a complete and documented operator: An annotated Java file and a single Markdown text file.

2.3 Support for User-Libraries

When being added to the RapidMiner plugins directory, the RapidMiner Beans library acts as a plugin that will search for external JAR files found in the `.RapidMiner5/beans` directory of the current user's home directory. Any JAR file within that directory will be checked for classes annotated with the `@OperatorInfo` annotation.

This alternate plugin mechanism has proven to be useful especially in environments where users are not allowed write-access to a central/shared installation of the RapidMiner suite.

3 Example – A simple Operator

In this section we will outline the implementation of a simple operator using the RapidMiner Beans library. As mentioned above, this requires the class to be annotated using the RapidMiner Beans annotations. Everything else follows the regular RapidMiner operator implementation.

We will outline the annotations in Section 3.1 and will show the use of parameter annotations in Section 3.2.

3.1 Operator Annotations

The sample code in Figure 1 implements a very basic RapidMiner operator. The class inherits the `Operator` super class and implements an almost empty `doWork()` method, which effectively passes through any input received at its input port.

Apart from the `@OperatorInfo` annotation, present in this class, the code follows the usual implementation of a RapidMiner operator. The annotation, however, defines all information that is required for registering an operator with RapidMiner. With the RapidMiner Beans library present in the Rapid-Miner plugins directory, all classes that are annotated with `@OperatorInfo` are added to the list of RapidMiner operators.

```
 1    import com.rapidminer.beans.annotations.OperatorInfo;
 2    import com.rapidminer.beans.annotations.ParameterInfo;
 3
 4    @OperatorInfo(name="Sample Operator", group="RapidMinerBeans.Example",
 5                  description="A simple pass-through Operator")
 6    public class SampleOperator extends Operator {
 7
 8        Inputport input = getInputPorts().createPort("input");
 9        OutputPort output = getOutputPorts().createPort( "output" );
10
11        /** Simply pass through the input object to the output port */
12        public void doWork(){
13           IOObject in = input.getDataOrNull();
14           if( in != null ){
15              output.deliver(in);
16           }
17        }
18    }
```

Figure 1: A simple RapidMiner operator. Lines 4-5 do provide the `@OperatorInfo` annotation, defining the name, group and tooltip-text of the operator within RapidMiner.

3.2 Adding Parameters with OperatorBeans

In this section we will be extending the operator implementation of Figure 1 by adding parameters following the JavaBeans convention. The parameters are annotated using the `@ParameterInfo` annotation of the RapidMiner Beans library. Figure 2 shows the enhanced example operator with additional parameters `lambda` and `name`. In the case of the `lambda` parameter in the example, the `@ParameterInfo` has been added to the class field of the operator. Alternatively, it is also possible to add this annotation to the `set`- or `get`-method as for the `name` parameter in Figure 2.

```
@OperatorInfo(name="Sample Operator", group="RapidMinerBeans.Example",
              description="A simple pass-through Operator")
public class SampleOperator extends OperatorBean {
    // port definitions left out for brevity...

    @ParameterInfo( required = true, min = 0.0, max = 1.0,
                    description = "A threshold value" )
    Double lambda = 0.0; // A parameter of type double

    String name = "";    // A string type parameter

    /** A set-method for the 'lambda' parameter */
    public void setLambda(Double d){
        this.lambda = d;
    }

    /** A set-method for the 'name' parameter */
    @ParameterInfo( required = false,
                    description = "An optional name parameter" )
    public void setName( String name ){
       this.name = name;
    }

    /** The doWork method */
    public void doWork(){
       if( lambda > 0.5 ){ ... }
    }
}
```

Figure 2: A `OperatorBean` example, which is a simple operator that provides JavaBean-style parameters. The OperatorBean class ensures that all parameters are set before the `doWork()` method is executed.

The annotations ensure, that the parameter lists of the operator can be detected. The RapidMiner Beans library also provides the necessary methods to inject the user-entered parameter values into the operator. For this, the class extends the super class `OperatorBean`, which ensures that all parameters have been set when the `doWork()` method is called during operator execution. The `OperatorBean` class also automatically provides the parameter type lists, required by RapidMiner to display the required parameters in the graphical user-interface. An analogue bean also exists for `OperatorChains`. Alternatively, the beans functionality is provided by the `RapidMinerBeans` class.

4 Writing Documentation

As mentioned before, the RapidMiner Beans library uses *Markdown* [2] as format for documentation. The documentation is provided by following a very simple convention:

> *If the operator class is `my.package.Operator`, the documentation is expected to be found in a file `/my/package/Operator.md` within the classpath.*

The objective of this simple rule is to encourage developers to document their operators by simply creating a single additional text file that provides a textual description.

4.1 The Markdown Format

The *Markdown* format used by the RapidMiner Beans library has been proposed by John Gruber and is a simple wiki-like dialect that may additionally integrate HTML tags for any elements not supported by Markdown. Markdown provides generic support for

- lists (numbered and bullet lists)
- emphasizing text (italic, bold)
- hyperlinks and images
- code blocks, block-quotes

Markdown has gained a lot of attention as the primary plain-text format for documentation at `github.com` and became widely accepted. Multiple tools like *pandoc* provide conversion support from markdown in various other formats.

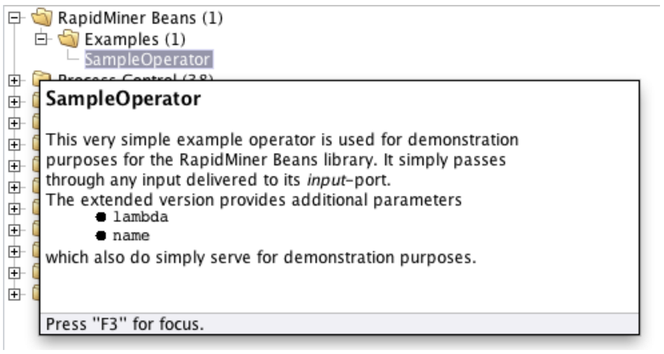
The RapidMiner Beans library uses MarkdownJ [3] to convert operator documentation to HTML which can then be displayed within RapidMiner. The following Figure 3 shows a sample of Markdown text describing the example operator introduced in Section 3.

```
The Sample Operator
===================

This very simple example operator is used for demonstration purposes
for the RapidMiner Beans library. It simply passes through any input
delivered to its *input*-port.

The extended version provides additional parameters

   - `lambda`
   - `name`

which also do simply serve for demonstration purposes.
```

Figure 3: Example markdown documentation for the example operator. The major headline is provided by underlining a line with equal characters. The * character switches to *italic* style, the back-ticks form `typewrite` code.

4.2 Documentation within RapidMiner

If operators are documented following the convention mentioned above, the RapidMiner Beans library automatically generates the required operator documentation at registration time of the corresponding annotated operator. This documentation is additionally enriched with the list of parameters created by RapidMiner. The parameter list documentation in turn is extracted from the parameter annotations described in Section 3.2.



SampleOperator

This very simple example operator is used for demonstration purposes for the RapidMiner Beans library. It simply passes through any input delivered to its *input*-port.
The extended version provides additional parameters
- `lambda`
- `name`

which also do simply serve for demonstration purposes.

Figure 4: Documentation of the *Sample Operator* operator, which has been written in the *Markdown* format.

5 Summary and Future Work

We presented the RapidMiner Beans library, which provides a simplified way to create and document new RapidMiner operators by using modern technologies such as Java annotations and JavaBean conventions. The plugin mechanism provided by the RapidMiner Beans library allows for automatically loading custom operators from plain JAR files available in a user's home directory. This is especially useful for developing and testing custom operators in environments where the central plugin directory is not writable by users.

For future work we are looking into automatically registering operators at runtime to support quick code-replacement of classes as is for example provided in hot-deployment settings such as Tomcat or JBoss. This will further shorten the development-cycle for new operators.

Another future direction is additional tool support to convert JAR files with annotated operator classes into regular RapidMiner plugins by generating the required XML description files from the annotated classes.

Acknowledgements This work was supported by the DFG within the Collaborative Research Center on *Providing Information by Resource-Constrained Data Analysis* (SFB-876).

References

[1] "JavaBeans API specification Version 1.01," 1997. http://download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/.

[2] J. Gruber, "Markdown." http://daringfireball.net/projects/markdown/.

[3] "markdownj – A Java port of Markdown, the text-to-html conversion tool." http://code.google.com/p/markdownj/.