Rapid Development of RapidMiner Extensions

Katharina Morik, Jan Czogalla
TU Dortmund University, Department of Computer Science
Joseph-von-Fraunhofer Str. 23, 44221 Dortmund, Germany
{katharina.morik}{jan.czogalla}@tu-dortmund.de

Abstract

Developing RapidMiner extensions is helpful for adapting existing toolboxes for one's needs. While using a simplified extension for teaching purposes, we encounter some practical obstacles that make debugging cumbersome. Hence, we introduce a RapidMiner extension that makes it easier for extension developers to integrate and test new features.

1 Introduction

RapidMiner offers opportunities to develop data mining processes without programming. It is one of the first interactive analytics programs. This is a most striking difference to other toolboxes such as MatLab or R. It allows users to rapidly apply data mining processes to their fields of interest, who are not programmers – "the masses" [2]. Data mining courses at universities may address students from business, science, or engineering, who do not intend to do their research in data analysis, but will use RapidMiner in order to gain insight into their data and the stories the data tells them [1].

Some courses, however, still need to teach in-depth knowledge about machine learning, data mining, or general analytics. These are for students, who will push the state of the art further and will become data scientists, data miners, or learning algorithm engineers. They are intended to carefully regard the links between theoretical results and algorithmic formalization. Actually, some problems that are hot topics in research only become clear after some experience with writing efficient programs. Hence, it is important that students implement their own learning algorithms, even if such implementations already exist. It will turn their attention to some practical problems, that have been solved in the last decades and they will appreciate these solutions.

Moreover, they would not forget the algorithms after the examination, because they have re-invented them on their own.

To get students started with data mining implementation exercises, the authors provide them with a RapidMiner extension with blank operators, i.e. operators that do nothing but have all constraints and frames needed to concentrate on implementing the specified algorithms. In particular, variants of top-down induction of decision trees, k-Means, FPgrowth, and lossy counting have been implemented by students using this extension. They could use all comfort of RapidMiner to read in data, preprocess them, run the algorithms and inspect results, while at the same time writing their own code. Overall, this has been very successful. However, operator development and testing are intertwined, and as a result students had to compile and package the extension multiple times to tweak their implementation. We see a deficiency in this process that all extension developers are facing.

1.1 The Problem of Development and Test Cycles

Students programming their own operators within RapidMiner by writing code into the prepared blank classes have to stick to the same cycle if they want to edit something:

1. close RapidMiner
2. search for the operator/class to be changed
3. edit the code
4. rebuild extension
5. restart RapidMiner

Besides the fact that rebuilding an extension and restarting RapidMiner costs some time, it is also unhandy. Since RapidMiner and its extensions are written in Java, there are several ways to monitor the execution of the running program. But even if RapidMiner is started in a debug mode and the relevant classes can be followed by this debugger, it is not possible to change or add method names and attributes in a class at runtime, due to the limitations of the Java Virtual Machine (JVM). This might be okay for fixing small bugs, but for testing parts of an operator again and again while developing it, or even adding new operators to an extension, this is much more time consuming than it has to be.

So how can we avoid this behaviour? Is there a way to rebuild extensions and deploy them anew in RapidMiner? At runtime? There certainly is. Each extension has its own Java class loader and if we were to replace this class loader and all objects loaded by it, we are easing development and testing.

In other environments, e.g. web deployment of Java Apps, there are already (proprietary) solutions like JRebel/LiveRebel[1] that address a similar problem. In this paper, we present our approach to this problem.

2 Supporting Extension Developers

Instead of the cycle shown above, we want another procedure of redeployment, i.e. we want to improve the process of applying changes of an extension to RapidMiner at runtime. This asks for a comfortable interaction of RapidMiner and the Integrated Development Environment (IDE). By IDE we refer to any program used by developers to create extensions and operators for RapidMiner. If a developer wants to change some operator behaviour, the following procedure is desired:

1. A right click on an operator opens a pop-up menu which contains an option to edit the source code.
2. The IDE comes to the foreground, showing the source code of that operator.
3. The developer edits the operator and saves changes.
4. The developer requests to redeploy the changed code using a menu.
5. After building the extension and redeploying it, RapidMiner comes to the foreground again.
6. The process runs again, and the developer is satisfied with the changes - or not and starts over again.

There are some more things to consider that users probably will not see, but are nevertheless important. To completely remove all (possibly faulty) objects created by a specific extension, it is essential to not only remove operators and their descriptions from a running RapidMiner application, but also to exchange any contribution made to the GUI and other enhancements.

Since not only students programming operators for learning purposes face these problems, but also extension developers in general, we have decided to pack this new functionality into a separate extension. We inspected the RaidMiner code and made some API modifications where they were necessary.

To model the desired behaviour, we need an extension for both RapidMiner and the IDE. At this point we decided to go for Eclipse[2] as the IDE to use, since RapidMiner itself is developed in Eclipse and therefore it comes with an

[1] www.zeroturnaround.com
[2] www.eclipse.org

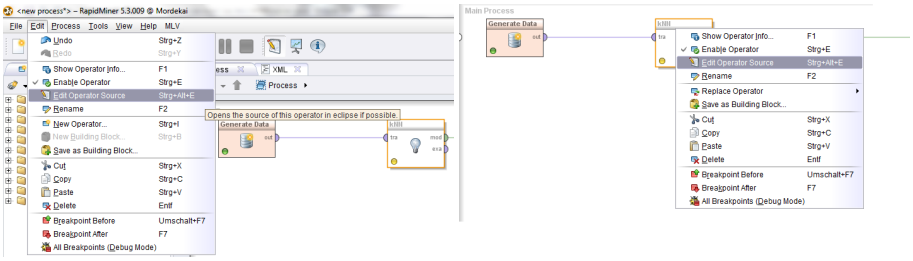

Figure 1: The edit source code action in the edit menu (left) and context menu of an operator (right)

Eclipse project file. Another reason is that among Java IDEs, Eclipse is very popular. This does not restrict our approach, because the most important part was to implement the redeploying process in RapidMiner, which is independent from the particular IDE used.

We will now discuss the steps we took to create these extensions for RapidMiner and Eclipse and will look at implementation details in the next subsections.

2.1 Extending RapidMiner for Extension Development

Some GUI components are needed to give developers control over this new redeploy cycle. The developer needs to edit or view the code of one particular operator. This requires some interprocess communication which is realized using Java RMI, the native Java implementation of Remote Method Invocation, which allows to use remote Java objects, e.g. in another JVM, in Java code as if they were local. This eases interprocess communication, although it adds some extra exception handling. The user can edit an operator from the already existing RapidMiner menus: you can either activate it from the main menu bar in the edit section or from the context menu of an operator or via the short cut Ctrl+Alt+E (see figure 1). RapidMiner then retrieves the class associated with the selected operator and tells the IDE via Java RMI to open the source code belonging to this class. If this is possible, RapidMiner is minimized to the task bar to bring the IDE to the foreground.

The functionality of redeploying is invisible to the RapidMiner user. It is invoked from the IDE. The communication from IDE to RapidMiner again works with Java RMI. It is possible to mark extension files (which are Java archives with the suffix "jar") for copying if the operating system blocks the current extension file and prevents replacing. To replace the modified extensions, some modifications to the *Plugin*-API had to be made.

Each RapidMiner extensions has an initialization class which has four

methods to initialize the extension. These methods are invoked during the start-up of RapidMiner to add GUI components, add special repositories and so on. For each of these extensions there will be a *Plugin* object at runtime, holding the id, name and initialization class of the extension among other information.

If an extension should be replaced, it first has to be completely removed from the RapidMiner context, allowing the JVM to clean up any outdated object. That implies that with the new version all the initialization methods need to be invoked again. Without removing everything that was added by the extension first, RapidMiner would have two versions available for everything this extension contributes.

For unregistering the extension, a method was added to the API that would remove that *Plugin* object from the context, and further methods were introduced to the initialization class that would take care of GUI components and other objects created and held by this extension. Those are so called tear-down methods, and their purpose is to remove the contributions made and leave RapidMiner as clean as if the extension was not installed. For these actions to succeed, developers have to take responsibility when they implement the initialization class by concentrating their attention to these tear-down methods as well as the initializing methods. When the *Plugin* object is removed from the context and the tear-down methods are done, the class loader of the extension is closed and the new version can be loaded from the very same jar file.

To re-register the modified extensions, *Plugin* objects now have two methods to initialize themselves after reloading, such that (i) all operators and descriptions can be loaded, and (ii) GUI elements and enhancements are loaded, so dependencies to other extensions can be fulfilled.

The IDE will deliver a list of modified extensions and our development extension then performs some additional steps for keeping RapidMiner consistent. The current RapidMiner process is saved and then removed from the context. Hence, changes made can take effect. Code that relies on a modified extension is marked to be redeployed too, thus keeping the dependency graph of the extensions consistent. After removing all modified and so marked classes and methods, the extensions whose jar files need copying are copied to the extension directory. Then, the extensions are recreated, initialized and registered with RapidMiner again. Finally, the process is reloaded with possibly new operators.

2.2 Modifying the IDE

For Eclipse, the extension mainly consists of internal listeners. We need to have a look at the projects in the current workspace, check if RapidMiner was started and if so wait for it to request to view the source code of an operator.

And of course we need a menu to give developers the opportunity to redeploy any modified extension.

For the extension to be active, there has to be the RapidMiner project and at least one RapidMiner extension project in the workspace. The listener checks on start-up if these conditions are fulfilled, but will make sure any newly added or modified projects are observed, to find out if they may be relevant. It also keeps track of the extensions that get changed and need to be redeployed.

If RapidMiner has been started from Eclipse, a listener waits for RapidMiner to send a class name via Java RMI to open the corresponding source code if possible. The specified class therefore must be present either in RapidMiner itself or in any extension in the workspace. If that is the case, the source code of the class is opened in a Java editor. If the class could not be found in the workspace, RapidMiner is told so and will produce an error dialog.

For the rebuilding of the modified extensions, a menu is created to let the developer decide, when to build and redeploy the extensions. As the extensions (and RapidMiner, too) are Ant projects[3], it takes some time to run the corresponding processes. If one of the observed extension projects was changed, it is stored in a separate list. For the purpose of rebuilding, this list is processed one by one due to the fact that only one Ant build can be run at a time. The Ant build for each extension will be run and checked for success. The only failure allowed is the final moving of the new jar file to the extension directory of RapidMiner. If this moving fails, the associated extension is marked for copying through Java RMI as mentioned before.

If RapidMiner is running after all extensions are built, Eclipse initiates the actual redeploying in RapidMiner via Java RMI, sending a list with the successfully rebuild extensions. If this was successful as well, RapidMiner is restored from its minimized state, hiding the IDE again and showing the reloaded process.

3 Conclusion

Even if RapidMiner is not executed, our program will take care of building extensions so that users do not have to build those themselves. As there can only be one Ant build in a JVM at a time, the most important gain of our approach is saving time and action needed to apply changes to RapidMiner at runtime. There is no longer a need to restart RapidMiner after changing or even adding some operators. Another advantage is to have a look into the RapidMiner source code. If you need to know what a particular RapidMiner core operator does, simply browse the source code via the "edit source code" action. This gives an easy access to the source code and is a further step into

[3] Apache Ant (www.ant.apache.org) is a scripting tool to e.g. build Java applications.

the direction of interactive data mining. Finally, our approach is the basis for loading new extensions installed via the marketplace directly into RapidMiner without a restart. This is supposed to speed up the sharing of operators in the RapidMiner community.

Acknowledgements The authors want to thank Rapid-I for merging the patch that made this development extension possible. Thanks also go to the students at LS8 who have helped to get rid of some quirks in the early stage of the extension.

References

[1] Markus Hofmann and Ralf Klinkenberg, editors. *RapidMiner 5 – Use Cases.* 2013.

[2] Matthew A. North. *Data Mining for the Masses.* Global Text Project Book, 2012.