USER MANUAL

Written by:
David Carlin
Jacob Kershaw
Clifford Black
Nick Faccenda
Bryan Nunez
Damen Tomassi

Table of Contents

1.0 Quick Guide

This document is intended for users with a basic understanding of coding and how drools files are set up. The basic knowledge of .drl files will allow the user to create new rules simply using the rule builder option.

Rules Engines

A rules engine is a software implementation of a Business Rules Management System. Such a system can provide an alternative computational model based on facts and actions. Rules are created with when/then conditions which act as traditional if/else statements in traditional programming. When a condition is met, its defined action is fired. The key aspect of a rules engine is the underlying algorithm which determines which rules to check first and proceeds to fire the actions in a logical manner. Drools, the rule engine used in this prototype, is based off of the ReteOO algorithm.

Program Description

This program is a prototype implementation of a rules based decision engine. As such, it is primarily intended for use as a researching tool in order to determine the viability of utilizing a rules engine in a Command and Control environment. Furthermore, the most effective way of running the prototype is via an integrated developer's environment such as Eclipse or IntelliJ.

Purpose

As a research tool, the prototype can simulate real-world situations based on the data sets fed into it. Ideally, users of the program are subject matter experts as knowledge on the individual data sets is required for rule building and execution. The prototype allows for users to utilize any csv file in their possession in order to create a set of rules and actions to take on its data.

Scope

The scope of this prototype is purely as a research tool. It is not intended for use in the field nor should it be implemented in mission critical systems.

2.0 System Requirements

2.1 Operating System Requirements

This prototype is designed to run on Mac, Windows, and Linux Operating systems.

2.2 Eclipse Requirements

This prototype recommends using Eclipse Mars 2, version 4.5.2.

2.3 Drools Requirements

This prototype requires Drools version 6.4.0 or later.

3.0 Key Features

As per the initial product specification document, this framework prototype is not coupled to a specific decision point and can be utilized within a Command and Control system. The versatility of the rules engine is such that disparate sets of data can be linked together for complex decision making. In order to satisfy the requirements of the framework, the following features have been made available to the user upon initiating the program:

3.1 Normalized Data Importing

The primary input to any rules engine are the facts with which the rules are to be configured for. In this case, a fact is the formatted data set which is parsed by the program and instantiated as a data object. The format of all data files is constrained to the comma separated value data type, though the design of the program allows for the implementation of other parsers as to add support for more data types. Another constraint of the data files is that the first column be dedicated as a timestamp, this is for the purpose of event simulation which will be detailed below.

3.2 Customized Rule Configuration

A rule engine functions by applying a set of rules to the given facts and determining what actions to take based on the observed data. There is no limit to the number of rules that may be inserted into the engine, making this functionality a powerful asset in real time decision making. This framework prototype allows for the configuration of a complex rule base that is customizable by the user and linked to the imported data set. A rule set is implemented as a .drl file, each newly created rule places another file into the working directory. Rules can also be modified after creation via a text editor within the program.

3.3 Customized Action Definition

The final piece of the puzzle to a rule engine framework are the corresponding actions to take should a rule's conditions be met. This prototype allows for the definition of two types of actions to take, the first being a simple output to the console and the second an *action object* which can then be reinserted into the rules engine for the chaining of multiple rules. Both the creation of rules and definition of actions take place within the same menu option of the prototype. The user is first required to specify the "when" statement of the rule, that is, what conditions to look for in the imported data. Afterwards, the "then" statement may be specified which is the resulting action.

3.4 Branched Decisions via Forward Chaining

In order to increase the complexity of the decision making process, it becomes necessary to allow for the chaining of rule executions to perform more robust actions. This implementation allows for the forward chaining of rules such that the fired actions of one rule are the conditions of another rule. A prerequisite to this tiered structure is the creation of an additional *action object* which is not tied to an imported dataset, but rather has user-created fields which a rule will then insert values for. Essentially, forward chaining is accomplished when Rule #1 sets the values of the action object, to which Rule #2 is checking for a specific value. If the action object is set to those values, Rule #2 can now be executed.

3.5 Event Simulation

The previous features were all in regards to the configuration process of the program. Once the user has imported all of the necessary data and created the desired rules, it is time to begin the program and see the rule execution in action. Previously, it has been stated that a constraint for the imported data is such that the first column of the csv file is the timestamp for the rest of the data. This is important because event simulation is essentially the data parser updating the appropriate data object with new values as it parses from row to row of the file. Upon each update, all inserted rules are checked to see if any conditions have been met. The simulation ends when all imported data files have been parsed completely.

4.0 Details on Functionality

4.1 Create a New Rule (Customized Rule Configuration)

This option is not accessible until the user imports at least one log file. Assuming the user already imported log file(s), this option first requests a filename. After entering a name for the file the user wants to make, it then gives the user some information and requests a package name (Use default rules for this prototype). Then the user is prompted to enter all objects he/she would like to import into the rule (if any). Next is entering the rule name, any string will be accepted here. Now the user needs to input ‘y’ if they are using new data, or ‘n’ if they are not using new data. Following that, the user is prompted to enter all objects they wish to use for this rule. Afterwards, the user is asked for the object to be evaluated and given a default option as well. Next is the field that is tied to the passed object, followed by three comparison operator options; equals, less than, or greater than, followed by the value desired, and then what to do as an output if the condition is met.

4.2 Select a LogFile (Normalized Data Importing)

The prototype is currently configured to handle csv files. The files must be contained in the correct folder. This option will prompt the user for a csv filename, which has to be located in the ‘Logs’ folder in this prototype. Afterwards, if found, it will alert the user with a success message. Otherwise, it will alert the user that the file was not found and ask for valid input or to type ‘EXIT’ to leave this option.

4.3 Select a folder containing multiple logfiles (Normalized Data Importing)

This option will prompt the user for a folder filename, which has to exist already in this prototype. Afterwards, if found, it will alert the user with a success message. Otherwise, it will alert the user that the file was not found and ask for valid input or to type ‘EXIT’ to leave this option.

4.4 Add new Data (Branched Decisions via Forward Chaining)

The fourth option is ‘Add new data.’ This option will prompt the user for the name of a new Data. After that, it asks the user for a number of fields, and then respective name for the amount entered. This option is intended for use in forward chaining applications. The newly created data can serve as an an object for which a set of rules will apply values for. Then, a second rule set can be created with conditional checks on the values of the new data and perform an action when the conditions are met.

4.5 Run through one line of the logfile (Event Simulation)

The first of the two menu options for event simulation. Both require that a log file has been imported and the rules in the system is nonzero. This option will update the dataObjects with the next line in the corresponding CSV files, and then fire all rules related to added or updated fields in those dataObjects.

4.6 Run through entire LogFile (Event Simulation)

The second of the two menu options for event simulation. Both require that a log file has been imported and the rules in the system is nonzero. This option will update the dataObjects with the next line in the corresponding CSV files, and then fire all rules related to added or updated fields in those dataObjects. This process will be repeated for every file in the LogFiles.

4.7 List all active rules (Customized Rule Configuration)

This option will display every rule that is placed in the current KieSession field ‘activeRules’. Each .drl filename will be printed out here.

4.8 List all inactive rules (Customized Rule Configuration)

This option will display every rule that is placed in the current KieSession field ‘inactiveRules’. Each .drl filename will be printed out here.

4.9 Toggle rules on/off (Customized Rule Configuration)

This option display all rules that are currently in the KieSession field ‘activeRules’, and requests the user to enter all rule names that he/she would like turned off. This can be done any amount of times until the user enters ‘done’. Then the KieSession refreshes and proceeds to display all rules that are currently in the KieSession field ‘inactiveRules’, and requests the user to enter all rule names that he/she would like turned on. This can be done any amount of times until the user enters ‘done’. Afterwards the user is returned to the main menu.

4.10 Open Rules Editor (Customized Rule Configuration)

This option is intended for expert users only. This will open a new window which is a standard text editor. The user will be able to custom write their .drl file(s) here and use the file>save option to save their creation as a new rule, or overwrite an older file. This tool will also allow the user to open already made rules and modify them. Afterwards, the user can use file>exit or the X button to close the editor and go back into the program.

4.11 Exit Program

This option will alert the user that the program has ended and close itself.

5.0 Simulation Details

5.1 Normalized Data

In order for this prototype to function, the data being brought in had to be normalized in some way for a consistent format. This is done through an XML-to-CSV conversion process, which will parse XML files and convert them into a format that this prototype can understand and interact with. This can be expanded to other formats in the future.

5.2 Configured rules

The rule builder is what takes care of most of this behind the scenes. Throughout the rule creation process (Section 4.1), the user is asked a series of questions that the program will then translate to properly formatted Drool's rules (.drl files). These rules are required to have a specific format so that all needed objects are imported, conditions are written properly, and actions are written out in a way that future rules can be dependent on them.

5.3 Logfiles

Logfiles are how the prototype stores its simulated data. These files consist of a header row, which is used as the key for all dataObjects made from each log, and beneath it corresponding values for each 'tick' of time that passes. This is used as a simulation of a live data feed.

5.4 Data Objects

DataObjects are a generic template the prototype uses to store raw data. It does this by using a hashmap, where a string is the key, and the corresponding value is the matching raw data. This object is utilized by using getters, setters, and several update methods for when new data is passed in.

5.5 Action Objects

Action Objects are a subclass of DataObjects, and are only created dynamically as a result of Rules. These objects are designed to help execute rule chaining. They function the same as DataObjects, with one different field called derived, which will allow the user to trace where the objects originated from.

5.6 Rule Chaining

Forward rule chaining is implemented into this prototype. This is done by a single, simple step. A rule must create an action object for chaining to occur. When this action object is created, it gets inserted into the KieSession, which then updates and fires all rules that use that action object as a part of a condition. This sequence can happen any number of times and is the base method of forward rule chaining.

6.0 Setup and Installation Guide

6.1 Eclipse for Developers

This is how a user will go about setting up Drools in Eclipse, and how to get this prototype into Eclipse for a developer to edit.

6.1.1 Installing Drools

This guide will assume the user have nothing related to Drools already installed onto the user's machine. This is going to be a simple version, and a more in-depth guide can be found here:
https://docs.jboss.org/drools/release/5.2.0.Final/droolsjbpm-introduction-docs/html/ch03.html

The first step is going to be installing GEF (Graphical Editing Framework). This is done by going to help > install new software and pasting http://download.eclipse.org/tools/gef/updates/releases/ into the 'work with' textbar. After the GEF option appears, check the box and select next, next, finish, and let it install. The close Eclipse.

The second step is installing drools itself. This does not have an automated option, so we begin by downloading the needed files from this link: http://www.jboss.org/drools/downloads.html. After the download completes, extract the files into your eclipse's main directory. When the extraction completes, open up Eclipse and you should be able to switch into the Drools perspective, or create a new Drools project.

The final step is to install the tools we need to utilize the features of Drools. Similar to step one, go to help > install new software and paste http://download.jboss.org/drools/release/6.4.0.Final/org.drools.updatesite/, check the box for 'Drools and jBPM' and select next, next, and finish. Restart Eclipse and the user is now ready to start creating and modifying Drools projects.

6.1.2 Bringing the Prototype into Eclipse

The first step is going to be to download the Prototype from this url, https://github.com/SirKersh/Walrus. Once download is complete, move the files into your Eclipse workspace.

The second step is to open Eclipse, file > new > project > Drools > Drools project. After that, select next, title the project Prototype_FINAL, and update the path to where the folder were moved to (being sure it ends with Prototype_FINAL). Now the user has the ability to modify or run this Prototype in a developer environment.