Title: Design documentation for an implementation of Drools	**Version:** 1.0

Software Design Document

Drools Implementation

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

Title: Design documentation for an implementation of Drools	**Version:** 1.0

Table of Contents

Title: Design documentation for an implementation of Drools	**Version:** 1.0

1 Introduction

1.1 Purpose

The purpose of this document is to outline the technical design of this implementation of Drools. It will be used to aid in the development of this software by providing details on its design. The intended readers of this document are future programmers who are to further work with this software.

1.2 Scope

The software that is to be produced is a program that, at its base, can take in external data, convert it into a format readable by Drools, and implement the actions indicated by the rules the user created. Further functionality includes the management of rules and data. The product is to be developed for ease of use by experienced and inexperienced users of Drools and Java.

1.3 Definitions, Acronyms, and Abbreviations

Drools - Drools is a business rule management system (BRMS) with a forward and backward chaining inference based rules engine, more correctly known as a production rule system, using an enhanced implementation of the Rete algorithm.

Business Rule Management System (BRMS) - is a software system used to define, deploy, execute, monitor and maintain the variety and complexity of decision logic that is used by operational systems within an organization or enterprise.

Rule - Rules are pieces of knowledge often expressed as, "When some conditions occur, then do some tasks."

.drl – Drools rule file format.

Production Memory - Rules are stored here.

Working Memory – Facts are stored here.

Agenda - A system with a large number of rules and facts may result in many rules being true for the same fact assertion; these rules are said to be in conflict. The agenda manages the execution order of conflicting rules.

KieServices - The KieServices is a thread-safe singleton acting as a hub giving access to the other Services provided by Kie.

KieFilesSystem (kfs) - KieFileSystem is an in memory file system used to programmatically define the resources composing a KieModule.

KieModule - A KieModule is a container of all the resources necessary to define a set of KieBases like a pom.xml defining its ReleaseId, a kmodule.xml file declaring the KieBases names and configurations together with all the KieSession that can be created from them and all the other files necessary to build the KieBases themselves.

KieContainer - A container for all the KieBases of a given KieModule.

KieBase -The KieBase is a repository of all the application's knowledge definitions. It will contain rules, processes, functions, type models. The KieBase itself does not contain runtime data, instead sessions are created from the KieBase in which data can be inserted and process instances started.

KieSession - A KieSession allows the application to establish an iterative conversation with the engine, where the state of the session is kept across invocations. The reasoning process may be triggered multiple times for the same set of data.

DataObject – Java class that contains data fields to be inserted into the drools rules engine for evaluation.

ActionObject – Type of DataObject that results from the creation of more data to be inserted into drools and then evaluated. This program's implementation of rule chaining.

LogFiles – Facts in CSV formats with the fact name as the header, and the value below it. Each row in a logfile will represent a "realtime" data record that will be used to create a DataObject.

1.4 Design Overview

The driver will provide a UI that will indicate the functions of the software. The user will choose an option from the list and the respective action will occur.

Facts will be updated by the parser and injected into the drools rules engine as DataObjects. For this implementation, facts will be obtained via CSV (logfiles) and their parsers, however, DataObjects provide methods to allow the addition of facts from different sources. An interface will be necessary for such.

DataObjects will store the parser that creates it in order for it to call on the parser to update the DataObject whenever new data is needed from the parser.

Rules will be managed by the RuleManager class. This includes the addition, removal, modification, and toggling of drl files. The RuleManager class will call on the KieManager to reflect these changes in the kfs, and to update the KieBase.

Rule chaining occurs when a rule creates new data fields. An ActionObject will be created with a map containing the field name as the key, and the data as the value. The ActionObject will then be injected into the current KieSession and a the rules that relate to the ActionObject will fire. The object will not be added to the KieBase, and therefore will not be a usuable data source for future KieSessions, unless created again through rule chaining.

A static KieBase is used to store the current prototype instance facts and rules. From this KieBase, KieSessions will be created and evaluated by Drools, then disposed. Multiple KieBases with different

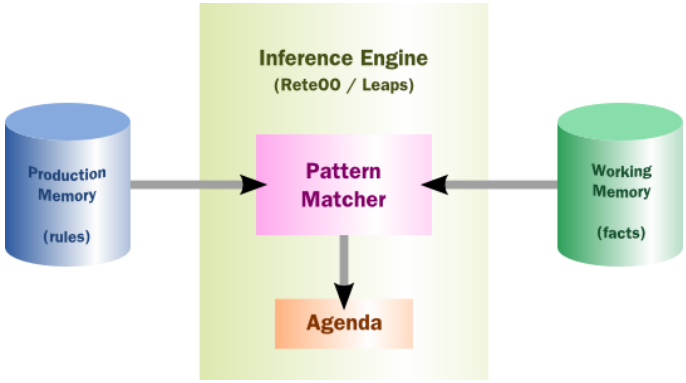
facts/rules could be used, however we decided it would be better to start with a single static KieBase, then update that KieBase everytime facts and/or rules change. This implementation uses a KieManager object to keep directly control the files in the KFS and rebuild it when necessary. Any addition of facts during the evaluation of rules will be added to the KieSession, and therefore will be handled by the currently running KieSession, and not the KieManager.

1.5 Versions Specifications

KIE 6.4.0 Final
Drools 6.5.0.Final
Java Version 8 Update 111

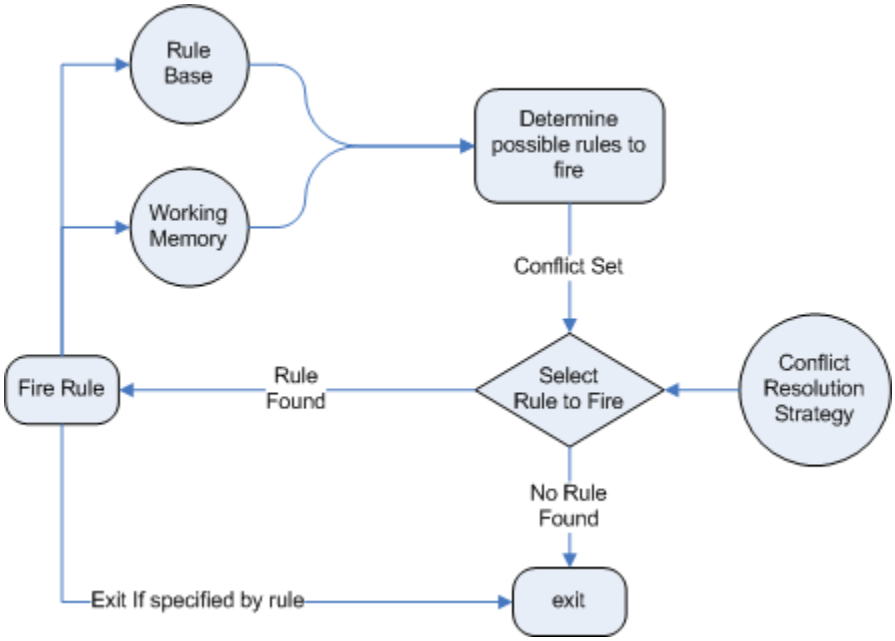
2 Drools

2.1 High-Level View of Rules Engine



Overview of the drools architecture. Facts will be asserted into working memory, which will result in one or more rules being true and to be scheduled for execution by the agenda.

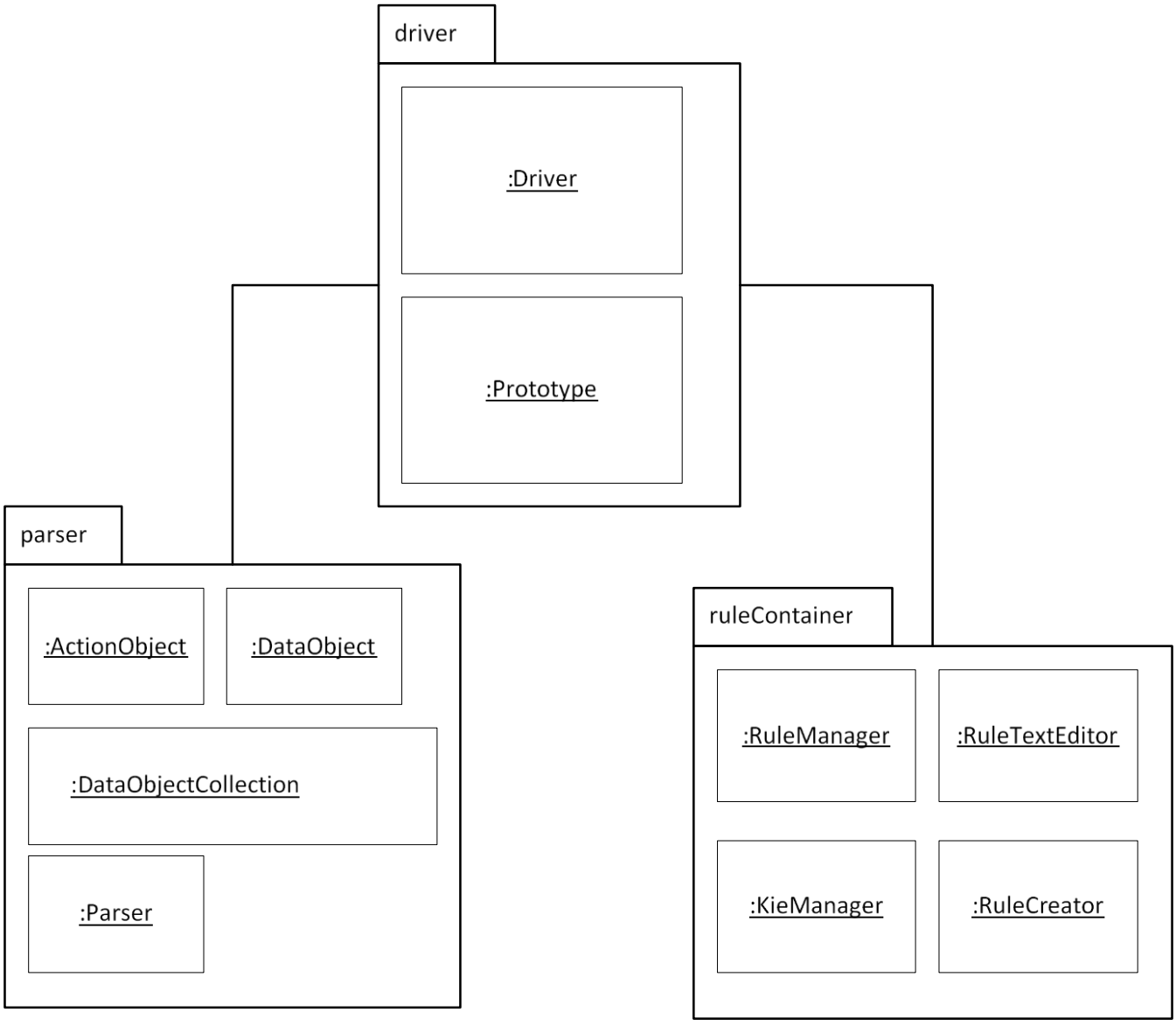
2.2 Rules Evaluation



Overview of the evaluation process provided by Drools.

3 Structural Models

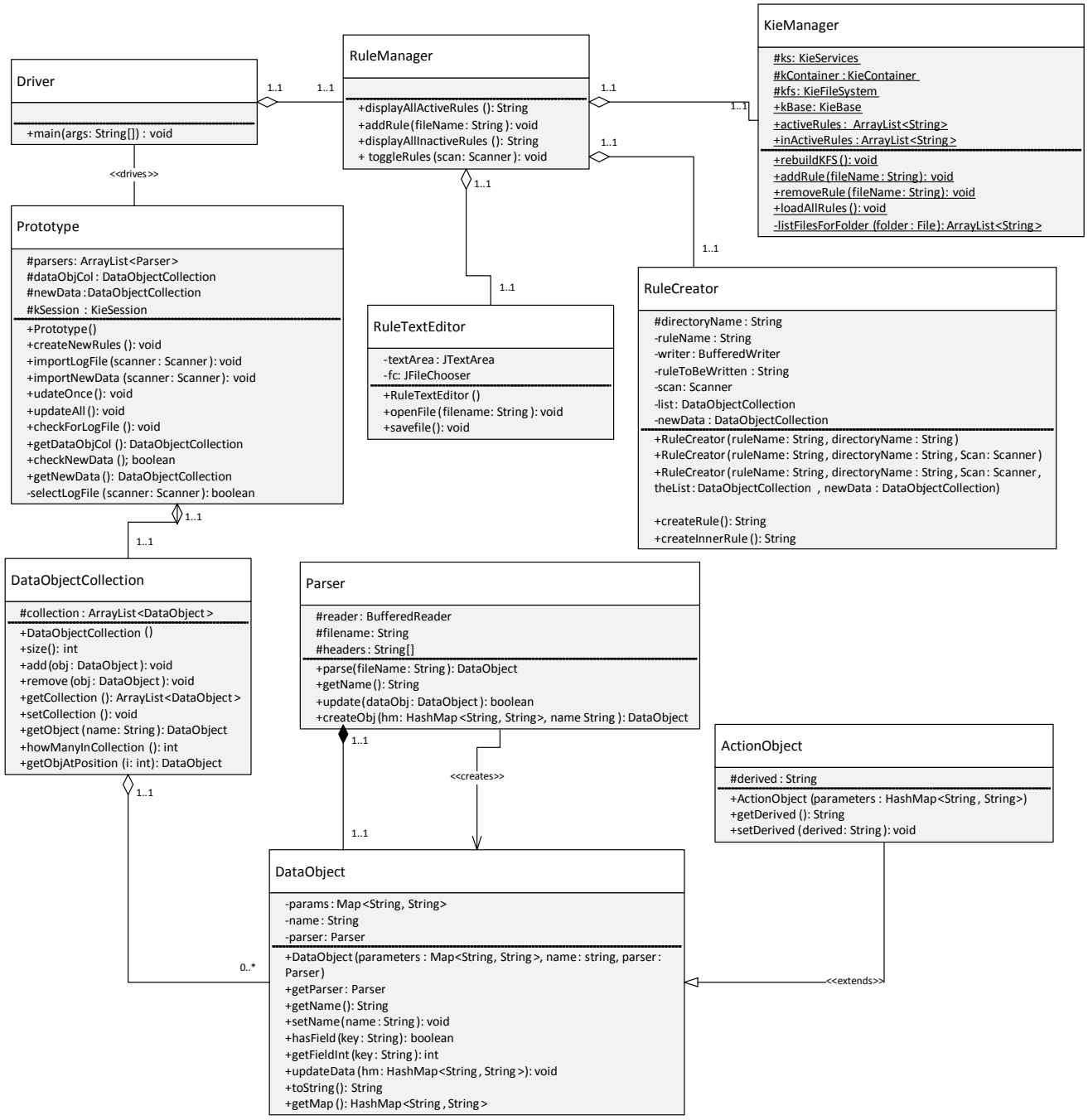
3.1 High Level Entities



High level view of all the entities in the system. The prototype will have access to the facts obtained by the parser, and the rules in the rule container. The prototype will then use drools to evaluate them.

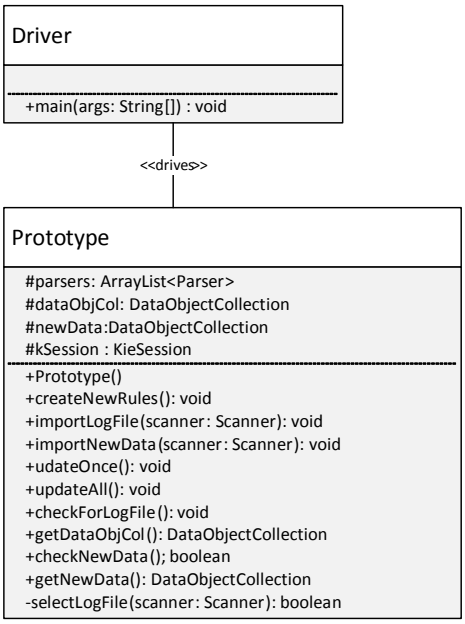
3.2 Class Diagrams

3.2.1 Drools Implementation Diagram



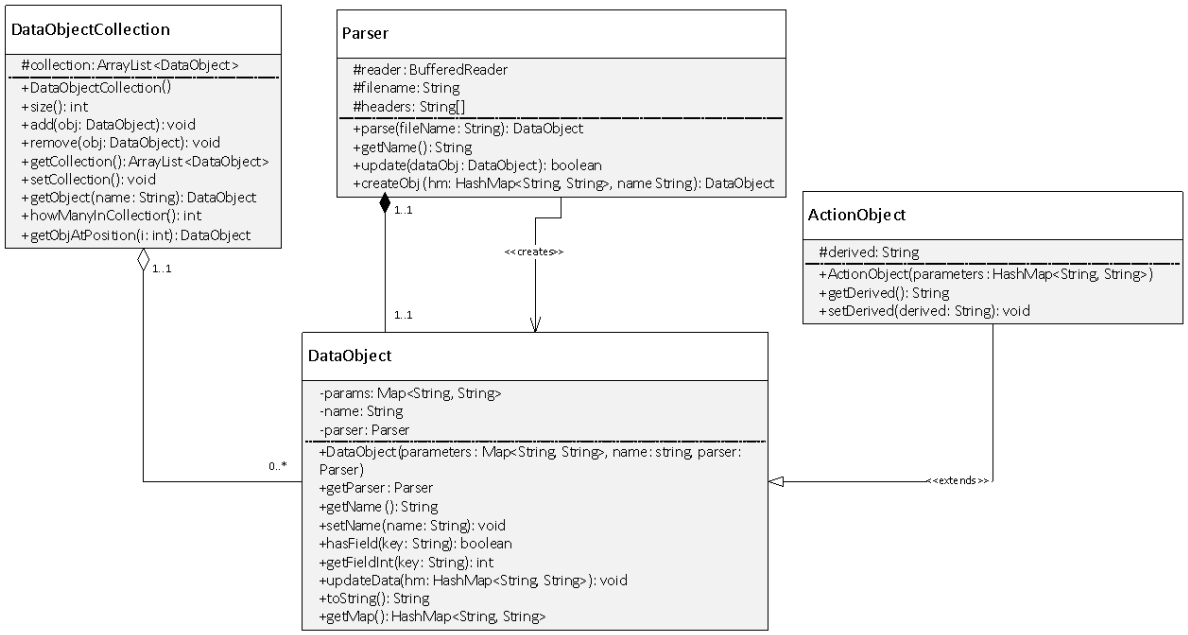
Overview of all the classes in the drools implementation prototype.

3.2.2 Prototype Interface Diagram



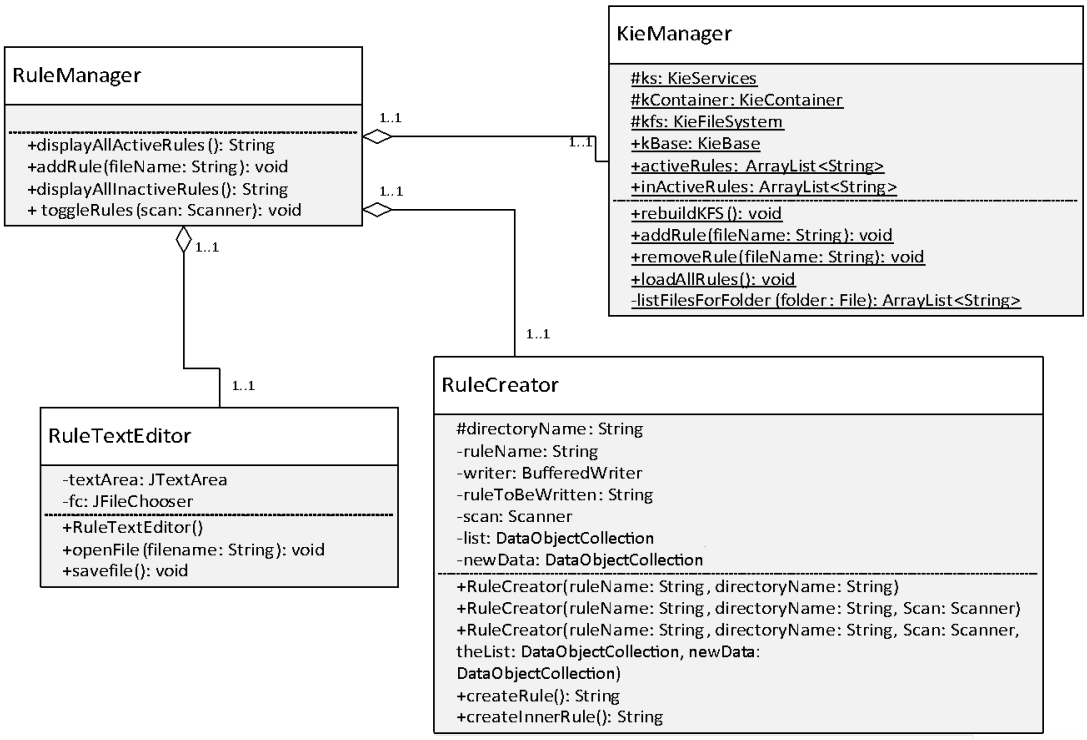
The drools implementation prototype interface provides the user with the functionality of the prototype.

3.2.3 Data Parsing Diagram



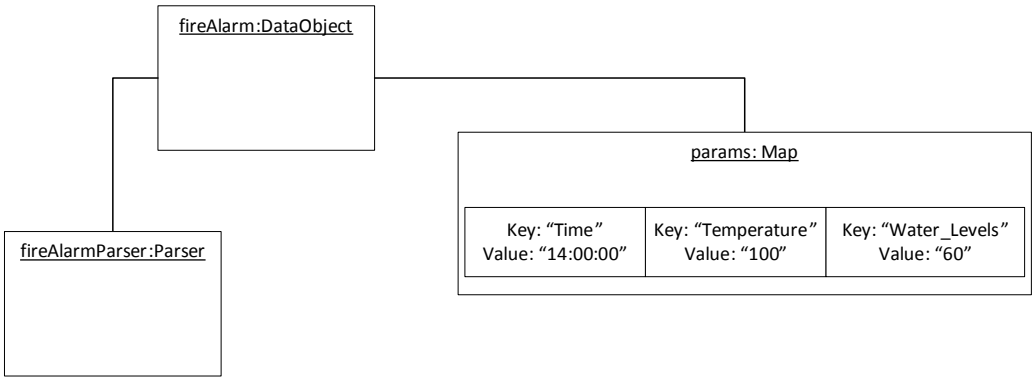
Data parser takes data from a CSV and places into the hashmap of a new DataObject to use as facts.

3.2.4 Rule Management Diagram



Rule management deals with creating, deleting, and modifying rules. It will have direct access to the physical files on the system the program is operating on. The KieManager will manage the rules when they become knowledge to the system.

3.3 DataObject Instance Diagram



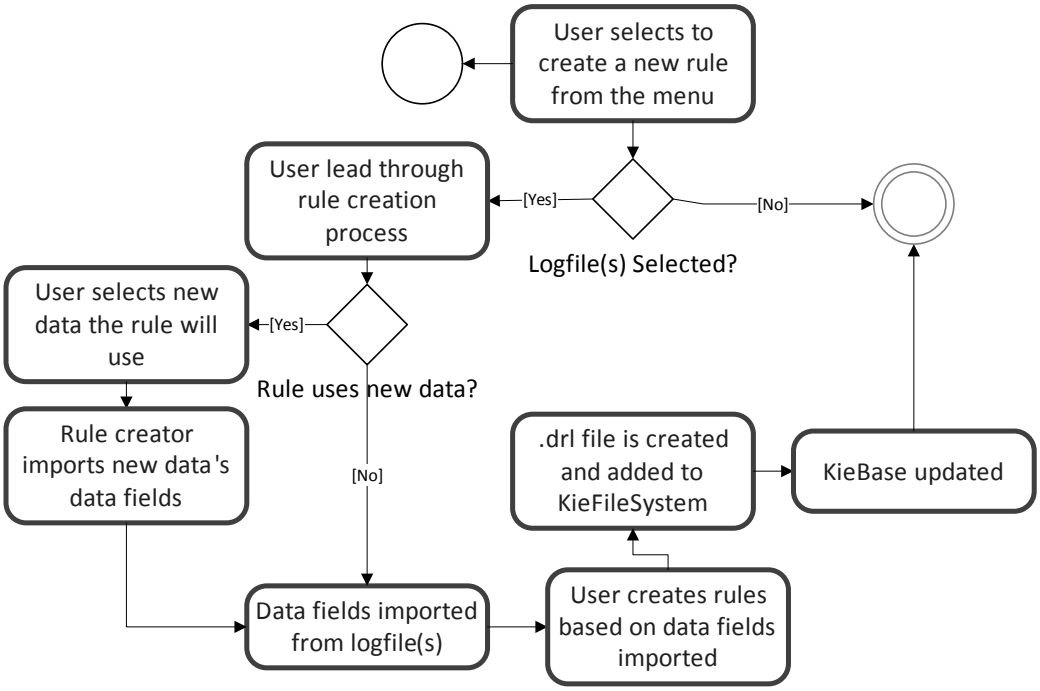
Time	Temperature	Water_Levels
13:55:00	120	70
14:00:00	**100**	**60**

The DataObject Instance diagram shows what fields a DataObject would contain. The hashmap will contain the facts that will be evaluated when it is added to drools and run. The values of the DataObject will change as the logfile is continued to be read, and the facts in focus change. You may have multiple DataObjects under evaluation at any given moment.

4 Behavioral Models

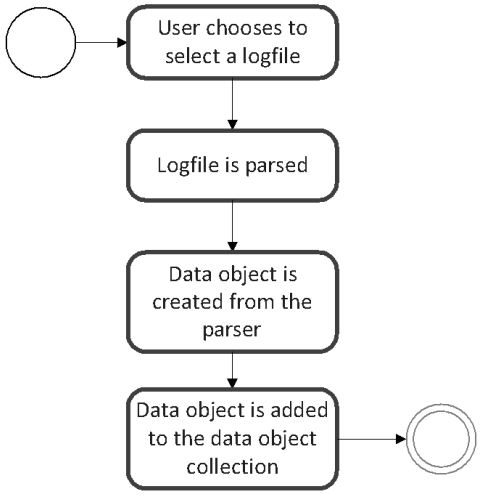
4.1 Activity Diagrams

4.1.1 Creating a New Rule



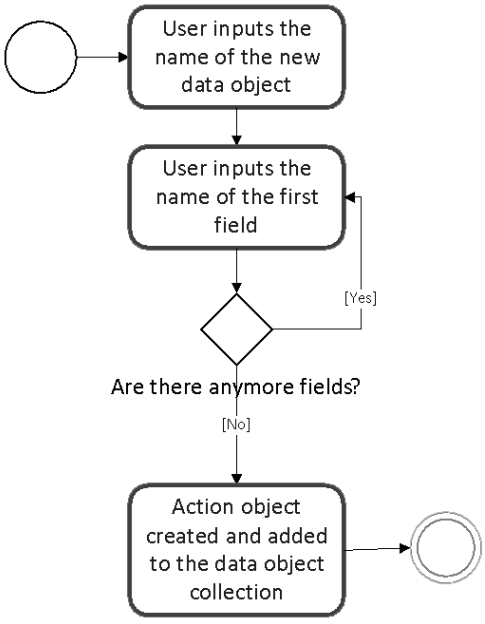
A user cannot create a new rule through the rule builder unless the applicable facts are known. Once the rule is created in the proper format, without errors, the drl created will be added to the kfs, and the KieBase will be updated.

4.1.2 Selecting a Logfile



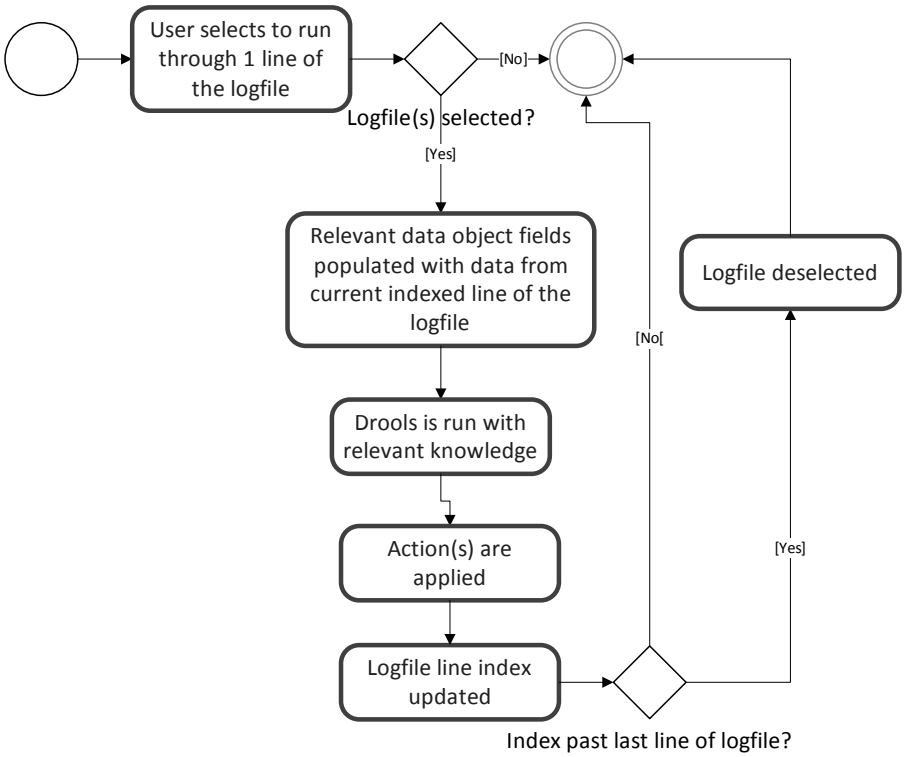
Selecting a Logfile creates the fields for the DataObject that will be used as facts for Drools. It will link the logfile to the parser, and the parser to the DataObject that is created.

4.1.3 Adding New Data



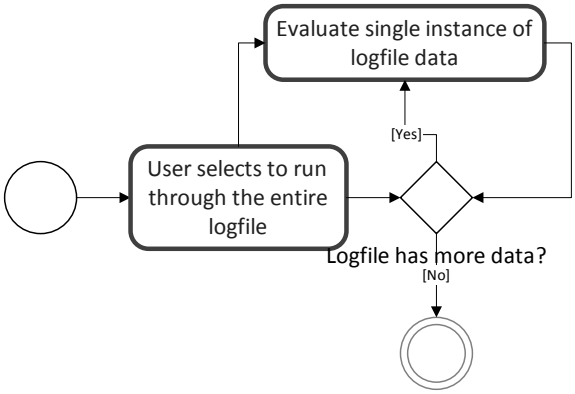
Adding new data is to let the prototype know that new data may or may not be created as a result of a rule(s) evaluating. The user is allowed to input as many possible fields they feel is necessary, however, duplicates are not allowed.

4.1.4 Evaluate Single "Realtime" Instance of Data from Logfile



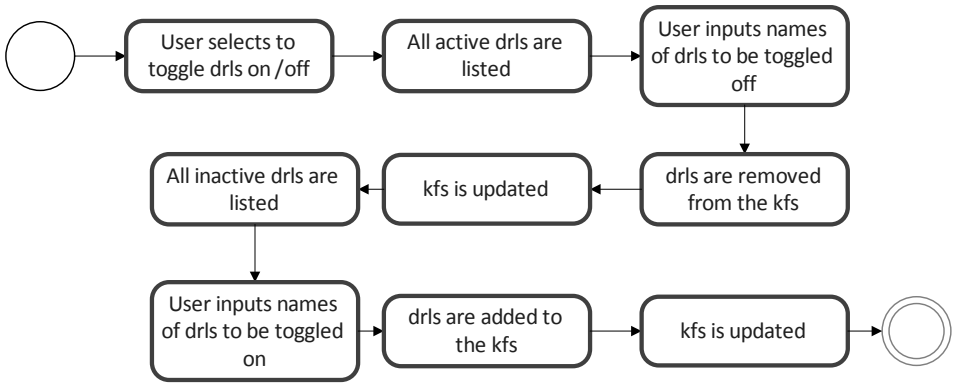
As long as the logfile(s) has(ve) more lines, the user may continue to select this option. DataObjects will be updated with their respective logfile's facts, and drools will proceed to evaluate them.

4.1.5 Evaluate Entire Logfile



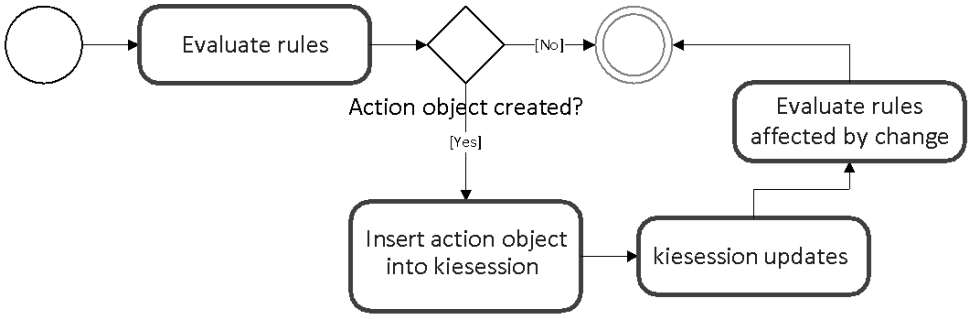
Loops through the entire logfile line by line, using the same process as described above.

4.1.6 Toggle Rules on/off



Toggling a rule off will remove it from the kfs, but it will still remain in the rules directory. They will not be evaluated when drools executes. Toggling a rule on will add it back to the kfs and cause it to be evaluated when drools executes.

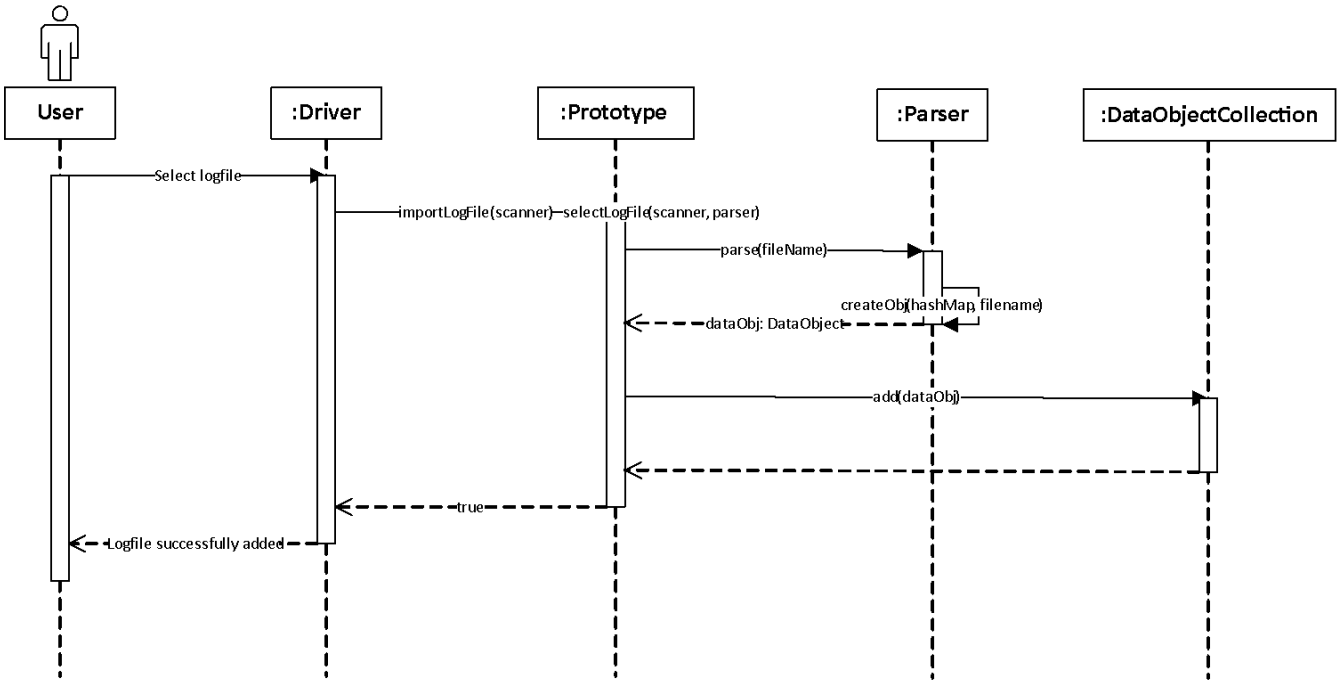
4.1.7 Rule Chaining



Rule chaining is exemplified by the use of an ActionObject. When a rule fires, it's evaluation may result in the creation of new data. This new data will be placed into a hashmap inside of an ActionObject. This ActionObject will then be placed into the working knowledge and evaluated. Any rules relevant to the facts in the newly created ActionObject will be evaluated.

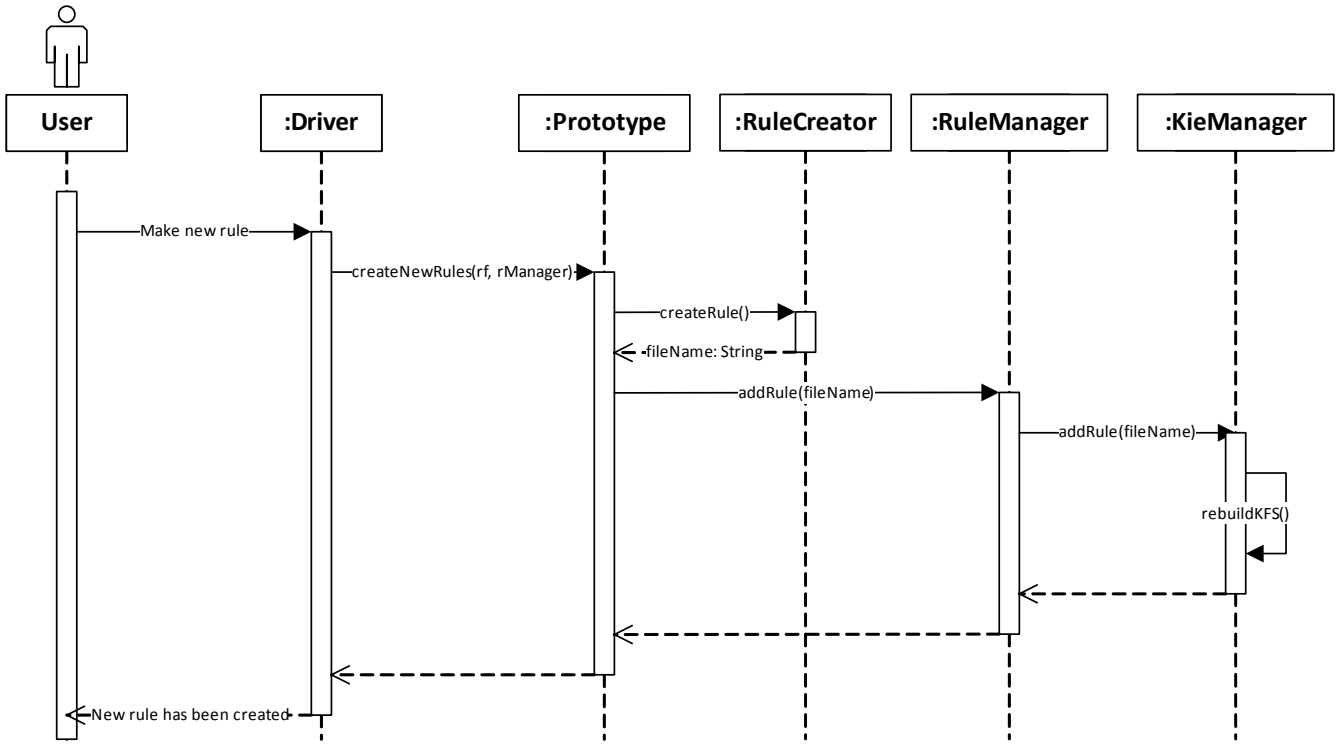
4.2 Sequence Diagrams

4.2.1 Adding Facts (via logfiles)



The user adds facts to the engine by selecting an applicable logfile. The parser will parse the logfile, create a DataObject based on the fields it has read in, then add the DataObject to the collection of DataObjects.

4.2.2 Adding Rules



Adding a rule requires the user go through the processes of creating a rule and ensuring that the format is correct. After that has finished, the ruleCreater will pass the filename of the new drl file that was created, back to the prototype. The prototype will then notify the ruleManager, which will then notify the KieManager that the new drl needs to be added to the kfs, and the kfs needs to be rebuilt.

5 Supporting References

Source	Title
https://docs.jboss.org/drools/release/5.3.0.Final/drools-expert-docs/html/ch01.html	Drools Documentation
https://docs.jboss.org/jbpm/v6.4/javadocs/overview-summary.html	KIE API