Event Processing Modes Walkthrough

Rules Project:

The ‘event-processing-mode-example’ in the ‘drools-example-rules’ repo contains a rule to demonstrate how a CEP system could detect and identify an example of a fraudulent trade event made by a single financial advisor. Our conditions that our rule is looking for that indicate a fraudulent trade event are two trades that adhere to the following conditions:

- Matching financial advisor
- Both trades are made on the same account
- The trades are made with identical stock symbols
- There is a time difference of 4 minutes or less between each trade event

event-processing-mode-example/src/main/resources/com/redhat/event_processing_mode_example/FinancialAdvisorFraudDetection.drl

```
package com.redhat.event_processing_mode_example.cloud_mode;

import java.lang.Number;
import tradeEvent.TradeEvent;
import report.Report;
import stockPriceEvent.StockPriceEvent;

rule "scenario1"
    dialect "mvel"
    when
        $trade1 : TradeEvent( $accountId : accountId, $financialAdvisorId :
        financialAdvisorId, $symbol : symbol, $trade1ID : id)
        TradeEvent( id != $trade1ID && accountId == $accountId &&
        financialAdvisorId == $financialAdvisorId && symbol == $symbol && this before [0s, 4m] $trade1, $trade2ID : id)

    then
        Report report = new Report();
        report.setCode( "Example project code, reason is too many trades by one financial advisor" );
        report.getTrades().add($trade1ID);
        report.getTrades().add($trade2ID);
        insert( report );
end
```

When this rule is fired and two *TradeEvents* are identified that match the pattern described in the rule, a new report will be generated with an example project code that would correspond to a financial advisor exceeding a threshold of trades, and the trade IDs that matched the rule are added to the Report. This setup allows for this rule to notify resources of the detected fraud event when the report is reviewed, but the system could also be configured to respond to this rule being matched in other ways, such as sending an alert to an investigation department, or a notification to the financial advisor that their recent activity was identified as suspicious.

Application Project:

In order for this rule to actually be triggered, we've set up Cucumber tests that test the rule using Cloud and Stream mode in the 'usecase-application' project in the 'drools-example-application'. These tests create *TradeEvents*, configure the *KieBase* to use either Cloud or Stream mode and query the *KieSession* for the *Reports* generated to make sure the rules fired correctly.

Below are the Cucumber feature files. Cucumber is a readable way to define a test and its data. Each 'Given', 'When', and 'Then' statement is mapped to pieces of Java Code in the 'CucumberSteps.java' file shown in the next few images below.

usecase-application/src/test/java/com/redhat/usecase/features/realtime_example/eventProcessingCloudModeExample.feature

```
Scenario: Testing a financial advisor making too many trades on the same account cloud mode
Given Trade Events:
|Id|Investor Id|Advisor Id| Price |Quantity|Symbol|Type    |  Timestamp          |
|1 | 1         |  1       | 20.00 | 10     | ANS  | Buy    |  27-09-1991 20:28:10 |
|2 | 1         |  1       | 20.00 | 11     | ANS  | Buy    |  27-09-1991 20:28:20 |
When I run the cloud mode example
Then I expect the following Reports to be created:
|Code                                                                  | Trades    |
|Example project code, reason is too many trades by one financial advisor | 1,2       |
```

usecase-application/src/test/java/com/redhat/usecase/features/realtime_example/eventProcessingStreamModeExample.feature

```
Scenario: Testing a financial advisor making too many trades on the same account stream mode
Given Trade Events:
|Id|Investor Id|Advisor Id| Price |Quantity|Symbol| Type   |
|1 | 1         |  1       | 20.00 | 10     | ANS  | Buy    |
|2 | 1         |  1       | 20.00 | 11     | ANS  | Buy    |
When I run the stream mode example
Then I expect the following Reports to be created:
|Code                                                                  | Trades   |
|Example project code, reason is too many trades by one financial advisor | 1,2      |
```

The ‘Given’ statement code parses the table of data below the statement in the feature file and creates a list of *TradeEvents*.

usecase-application/src/test/java/com/redhat/usecase/CucumberSteps.java

```
@Given("^Trade Events:$")
public void trade_events(DataTable table) throws Throwable {

    for (Map<String, String> row : table.asMaps(String.class, String.class)) {
        TradeEvent trade = new TradeEvent();

        trade.setId(row.get("Id"));
        trade.setAccountId(row.get("Investor Id"));
        trade.setFinancialAdvisorId(row.get("Advisor Id"));
        trade.setPrice(Double.parseDouble(row.get("Price")));
        trade.setQuantity(Integer.parseInt(row.get("Quantity")));
        trade.setSymbol(row.get("Symbol"));
        trade.setType(row.get("Type"));

        if (row.get("Timestamp") != null) {

            trade.setTimestamp(parseDate(row.get("Timestamp")));

        }

        trades.add(trade);

    }
}
```

Below is the ‘When’ statement code for both tests. Each calls ‘fireAllRules’ in the *ReportService* class. This function in the class takes in the artifactId of the Maven Project the rule belongs to, the *TradeEvent* list created in the ‘When’ statement, the processing mode type, and the session clock type.

usecase-application/src/test/java/com/redhat/usecase/CucumberSteps.java

```
@When("^I run the stream mode example$")
public void i_run_the_stream_mode_example_rules() throws Throwable {
    // Map a business name to the artifactId
    reportsGenerated = reportService.fireAllRules("event-processing-mode-example", trades, "stream", "realtime");
}
```

```
@When("^I run the cloud mode example$")
public void i_run_the_cloud_mode_example_rules() throws Throwable {
    // Map a business name to the artifactId
    reportsGenerated = reportService.fireAllRules("event-processing-mode-example", trades, "cloud", "realtime");
}
```

The ‘fireAllRules’ function uses another class called *BRMSUtil* to created a *KieBase* using the processing mode and artifactId passed in as parameters. It then creates a Stateful *KieSession* from that *KieBase* that uses the session clock type passed in as a parameter. It then adds all the *TradeEvents* to the session, fires all rules, and queries for the reports generated.

usecase-application/src/main/java/com/redhat/usecase/services/ReportService.java

```
public ArrayList<Report> fireAllRules(String artifactId, ArrayList<TradeEvent> trades, String mode,
        String clockType) {

    ArrayList<Report> reports = new ArrayList<Report>();
    TrackingAgendaEventListener listener = new TrackingAgendaEventListener();

    BRMSUtil brmsUtil = new BRMSUtil(artifactId, mode);
    KieSession ksession = brmsUtil.getStatefulSession(clockType);

    // ksession.addEventListener(listener);

    for (TradeEvent trade : trades) {

        ksession.insert(trade);
        if (clockType.equals("pseudo")) {
            SessionPseudoClock clock = ksession.getSessionClock();
            clock.advanceTime(6, TimeUnit.MINUTES);
        }
    }
    try {
        ksession.fireAllRules();
    } finally {
        QueryResults queryResults = (QueryResults) ksession.getQueryResults("reports generated");

        // System.out.println(listener.getMatchList());

        for (QueryResultsRow row : queryResults) {
            Report report = (Report) row.get("$report");
            reports.add(report);
        }
        if (ksession != null) {

            ksession.dispose();
        }
    }
    return reports;
}
```

usecase-application/src/main/java/com/redhat/usecase/utile/BRMSUtil.java

```
public BRMSUtil(String artifactId, String mode) {

    KieServices kServices = KieServices.Factory.get();

    ReleaseId releaseId = kServices.newReleaseId("com.redhat", artifactId, "LATEST");
    kContainer = kServices.newKieContainer(releaseId);

    // configure process mode type
    KieBaseConfiguration kBaseConfig = KieServices.Factory.get().newKieBaseConfiguration();
    if (mode.equals("stream")) {
        kBaseConfig.setOption(EventProcessingOption.STREAM);
    } else {
        kBaseConfig.setOption(EventProcessingOption.CLOUD);
    }

    kBase = kContainer.newKieBase(kBaseConfig);

    KieScanner kScanner = kServices.newKieScanner(kContainer);

}
```

For the Stream Mode test we are using the real-time session clock. The Stream mode test needs the timestamp to be updated in the *TradeEvent* upon insertion for it to work as a real time event processing usecase. The timestamp of a *TradeEvent* will only be updated with the session clock time upon insert if the timestamp attribute is not mapped to a field on the *TradeEvent* object. In the current domain model for these projects we do have the timestamp attribute mapped to a field so we need to comment that out in order for these tests and our rule

to work. The domain model is in the ‘usecase-domain’ project in the ‘drools-example domain’ repo. You will need to comment out the line of code above the *TradeEvent* class declaration.

usecase-domain/src/main/java/tradeEvent/TradeEvent.java

```
@org.kie.api.definition.type.Role(org.kie.api.definition.type.Role.Type.EVENT)
@org.kie.api.definition.type.Timestamp("timestamp")

public class TradeEvent implements java.io.Serializable {
    private String accountId;
    private String financialAdvisorId;
    private Double price;
    private int quantity;
    private String symbol;
    private String type;
    private Date timestamp;
    private String id;
```

Once the ‘fireAllRules’ function in the *ReportService* class is done it will return all the *Reports* created in the *KieSession*. The ‘Then’ statement code compares the *Reports* returned with the *Reports* expected (parsed from the feature file).

usecase-application/src/test/java/com/redhat/usecase/CucumberSteps.java

```
@Then("^I expect the following Reports to be created:$")
public void i_expect_the_following_Report_to_be_created(DataTable table) throws Throwable {
    ArrayList<Report> reports = new ArrayList<Report>();
    for (Map<String, String> row : table.asMaps(String.class, String.class)) {
        Report report = new Report();
        report.setCode(row.get("Code"));

        ArrayList<String> list = new ArrayList<String>(Arrays.asList(row.get("Trades").split(",")));
        report.setTrades(list);
        reports.add(report);
    }

    Assert.assertTrue(arraysMatch(reports, reportsGenerated));

}
```

You can run this feature files by locating *src/test/com/redhat/usecases/runners/RunCloudExample.java* and *src/test/com/redhat/usecases/runners/RunStreamExample.java* These can be run as Junit Tests in your IDE.