UNIVERSITÀ DI PISA

MSc in Computer Engineering

Software System Engineering

Secure Pos

TEAM MEMBERS:

Fabrizio Lanzillo

Federico Montini

Federica Perrone

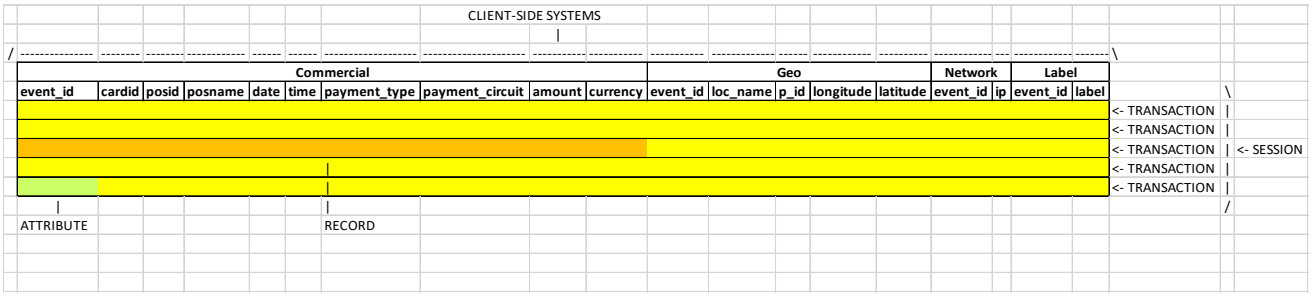
Riccardo Sagramoni

Veronica Torraca

Academic Year 2022/2023

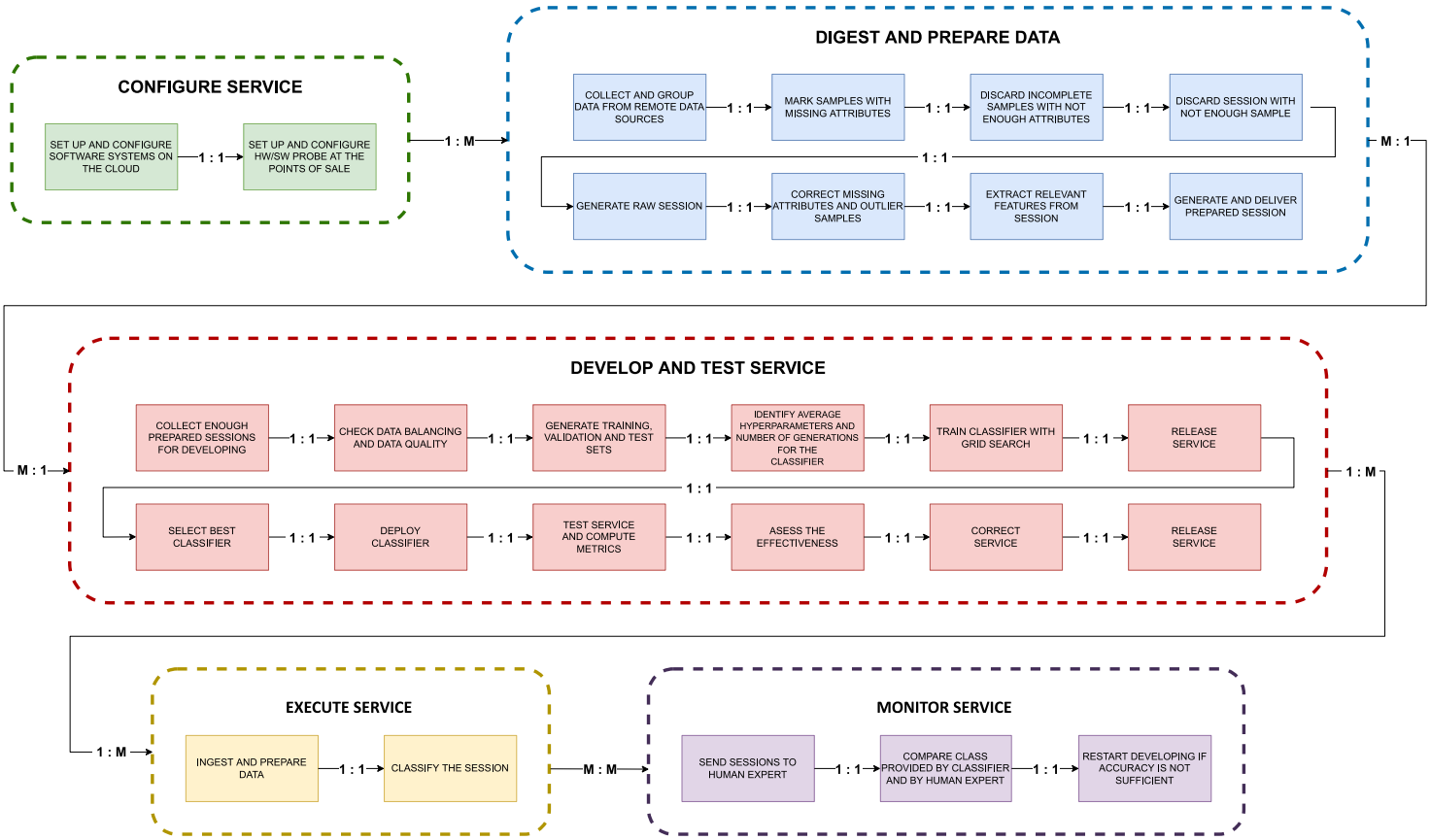
Summary

Glossary



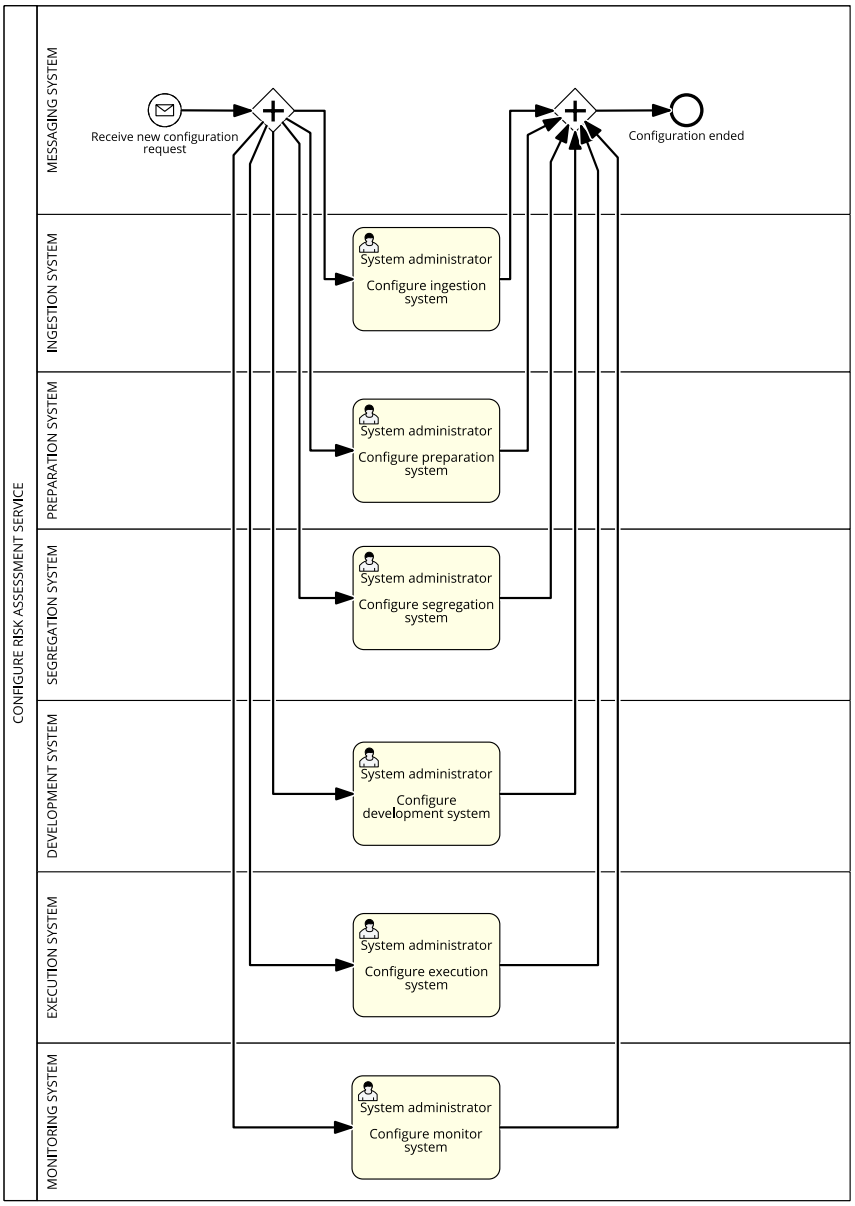
TERM	DEFINITION
Session	Set of **transactions** sent to the toolchain. The classifier will classify one session at a time.
Transaction	Set of all the data provided by the client-side systems and concerning a single payment.
Record	Data concerning a single transaction from a single client-side system.
Attribute	Single information relative to a specific record. A record is composed of multiple attributes, which represents the data provided by the client-side system.
Client-side systems	Four independent systems that provide the data.

Workflow Requirements: Process Landscape Diagram

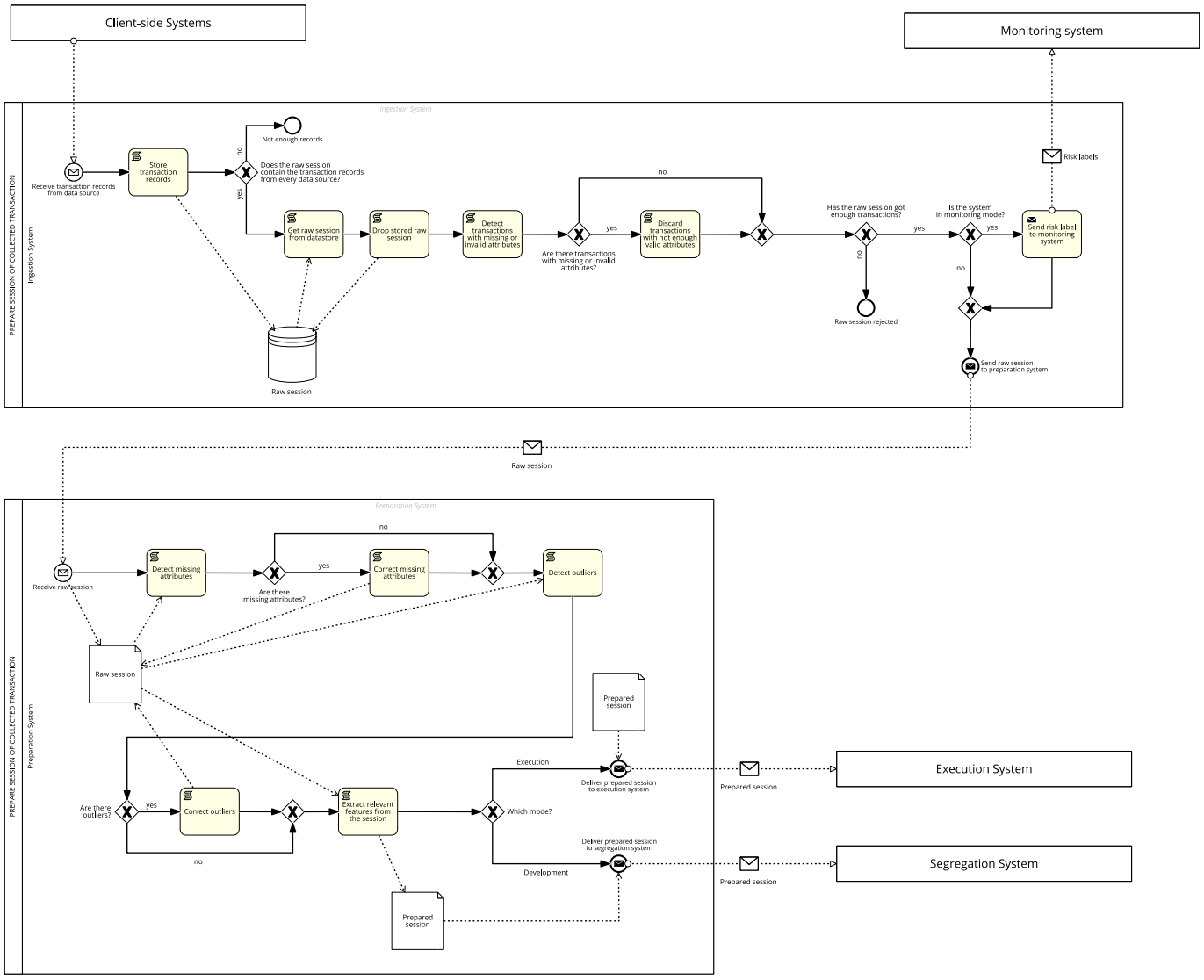


Workflow Requirements: BPMN Diagrams

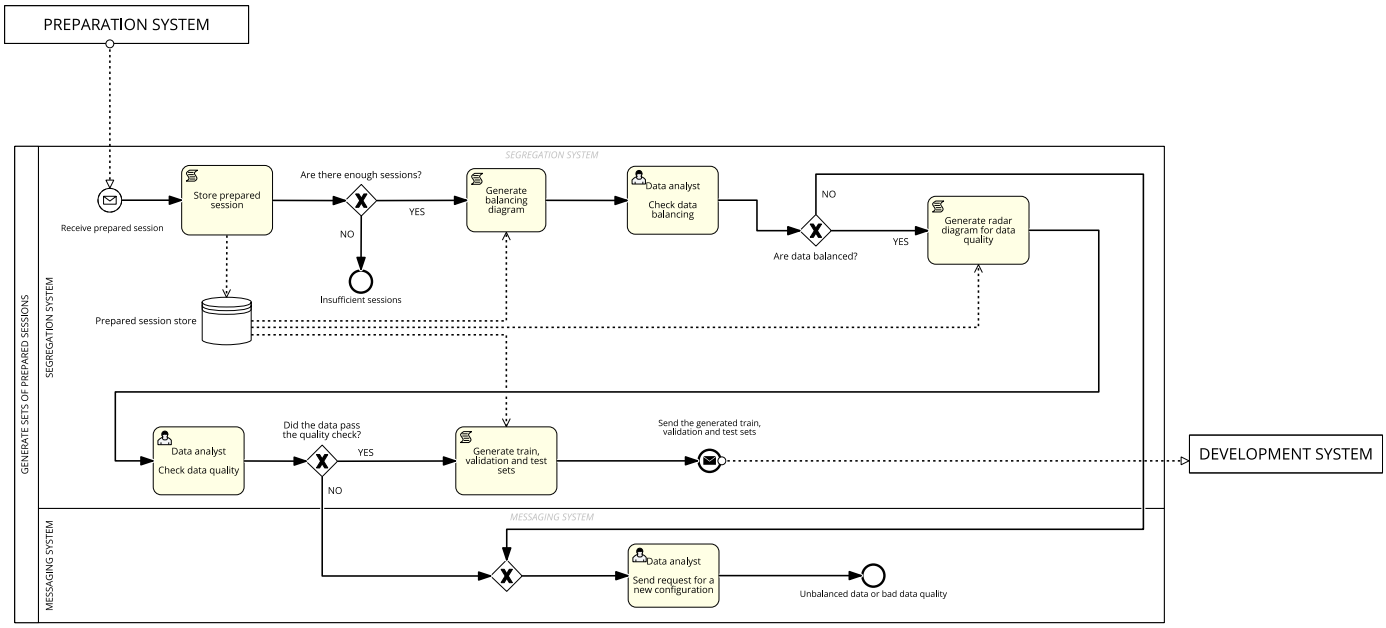
Configure Risk Assessment Service



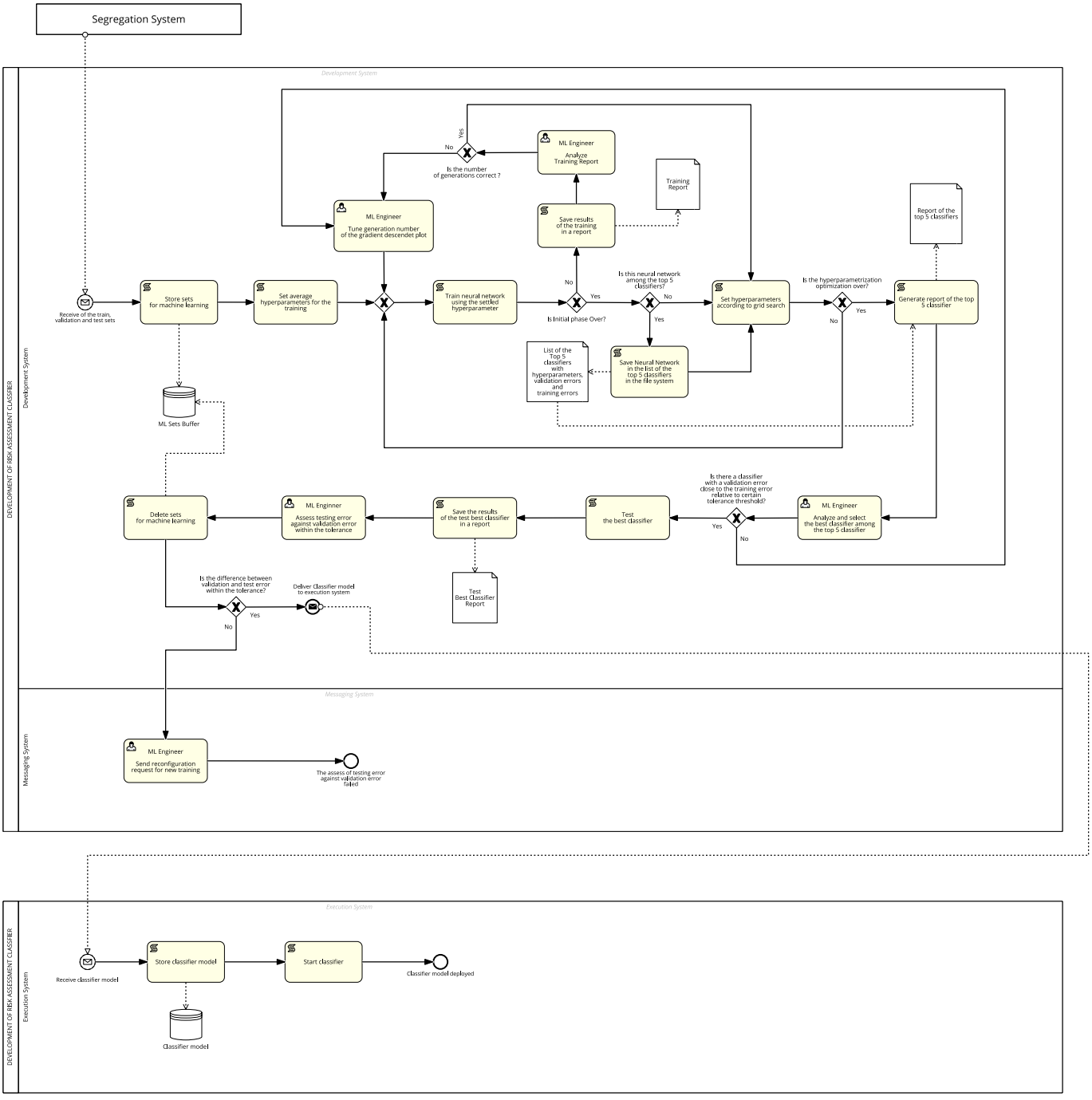
Prepare Session of Collected Transaction



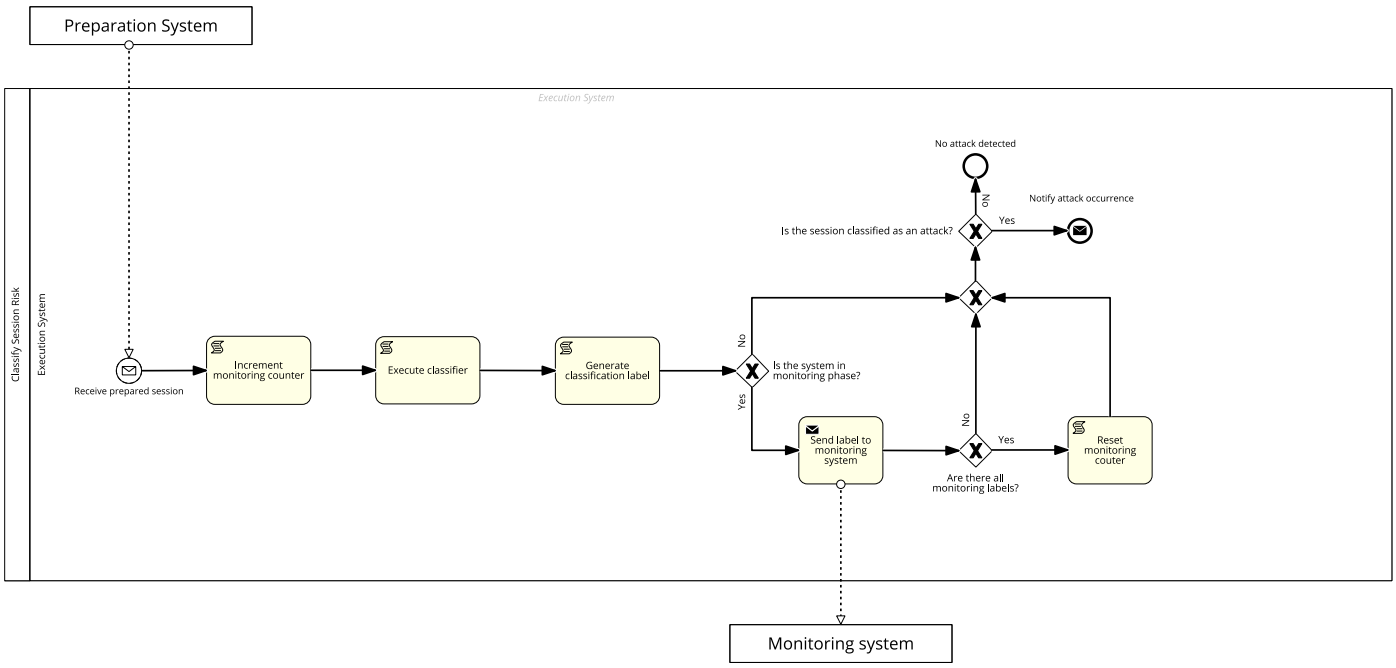
Generate Sets of Prepared Sessions



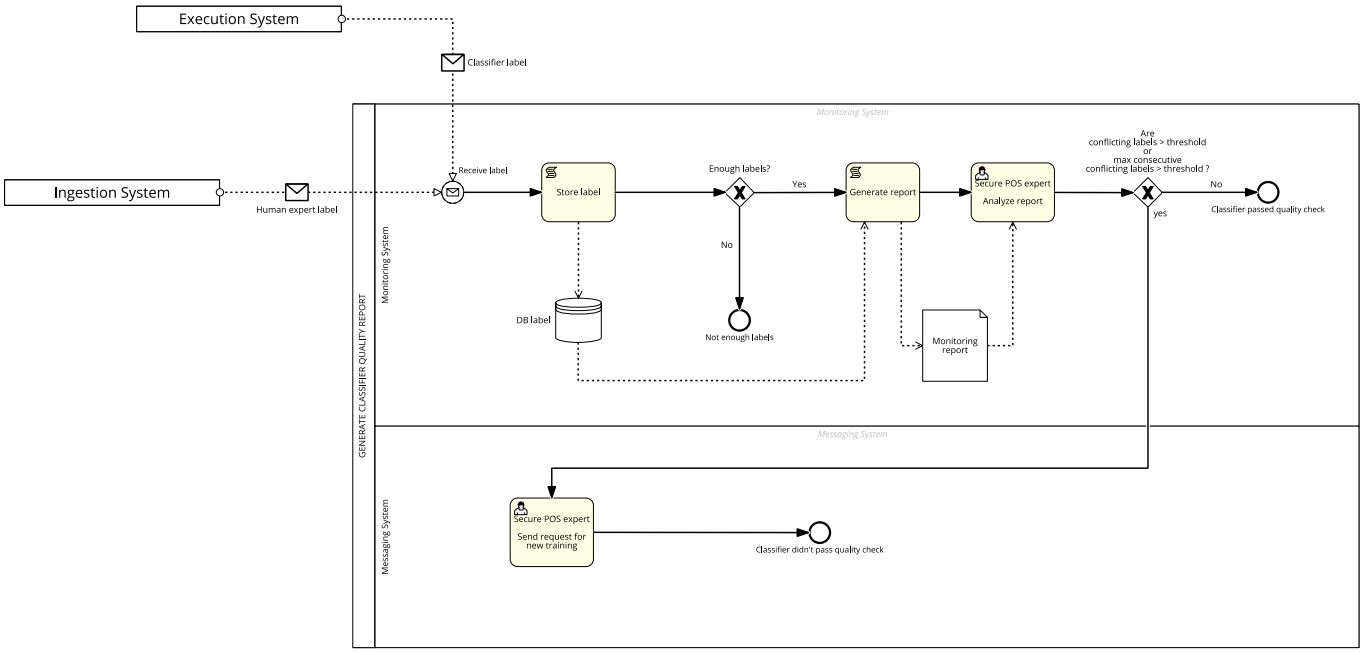
Development Of Risk Assessment Classifier



Classify Session Risk

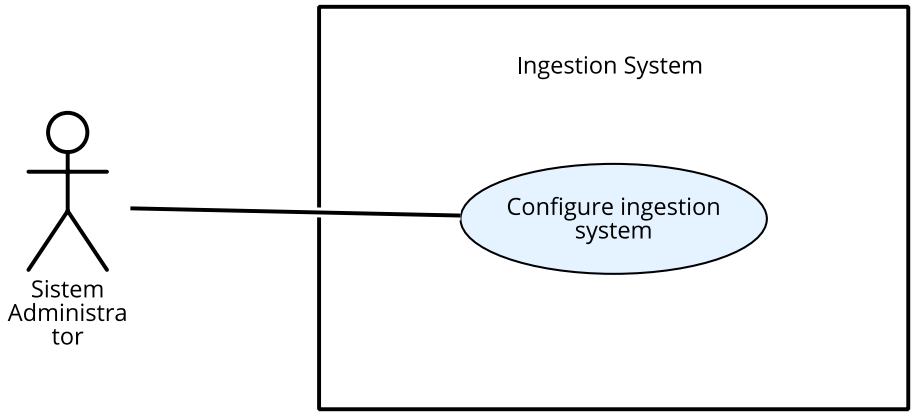


Generate Classifier Quality Report

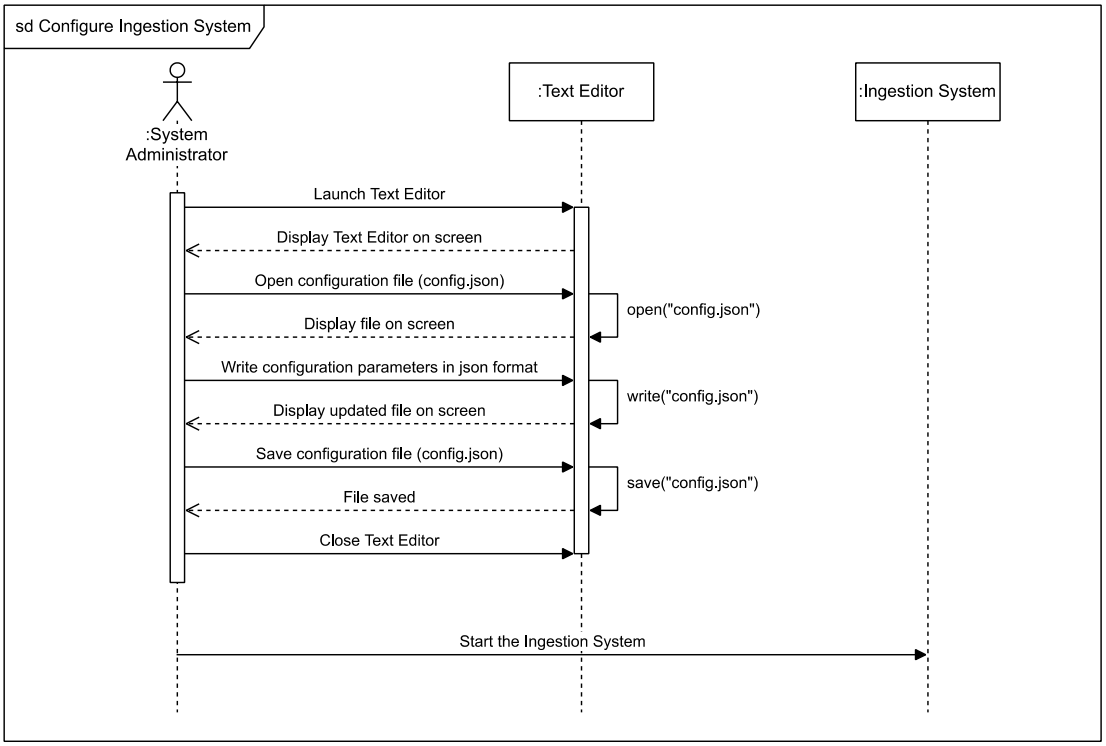


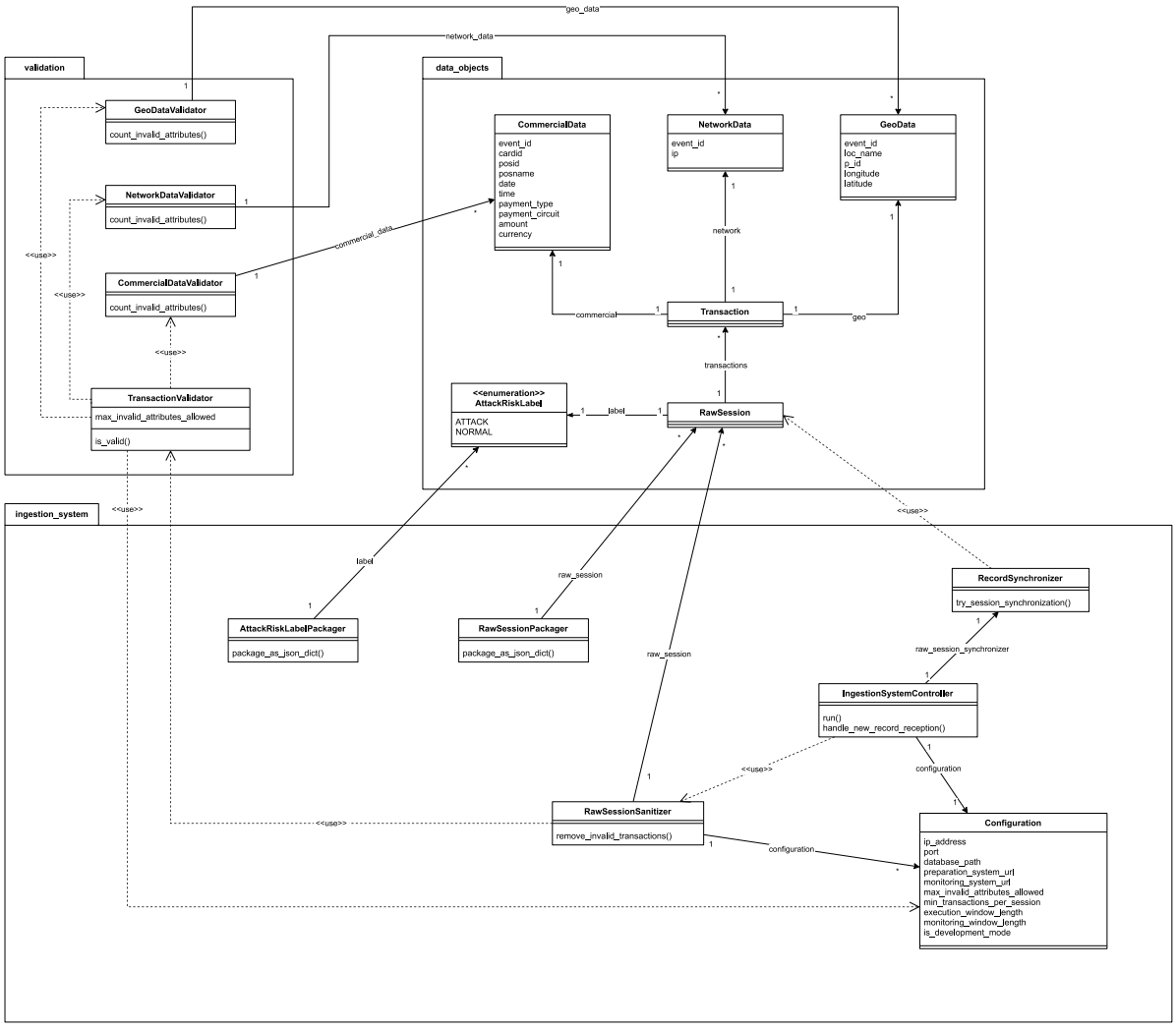
Workflow Analysis

Ingestion System

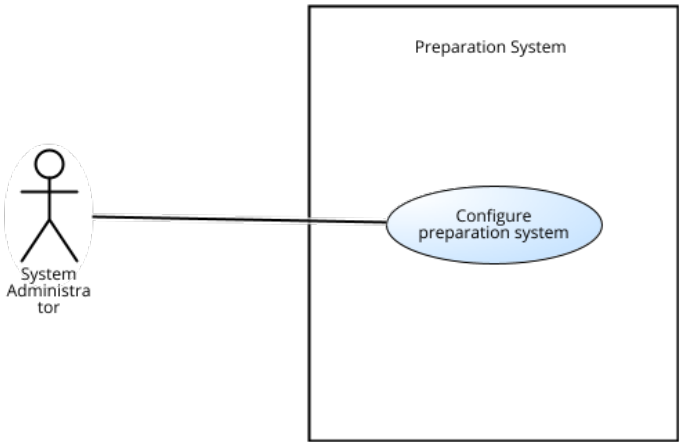


Configure Ingestion System
ID: UC0
Actors: System Administrator
Preconditions: 1. A new configuration for the Ingestion System is requested.
Flow of events: 1. The System Administrator launches the Text Editor. 2. The System Administrator opens the configuration file. 3. The System Administrator writes the configuration parameters in json format. 4. The System Administrator saves the configuration file. 5. The System Administrator closes the Text Editor. 6. The System Administrator starts the Ingestion System.
Postconditions: 1. The Ingestion System is configured. 2. The Ingestion System has started

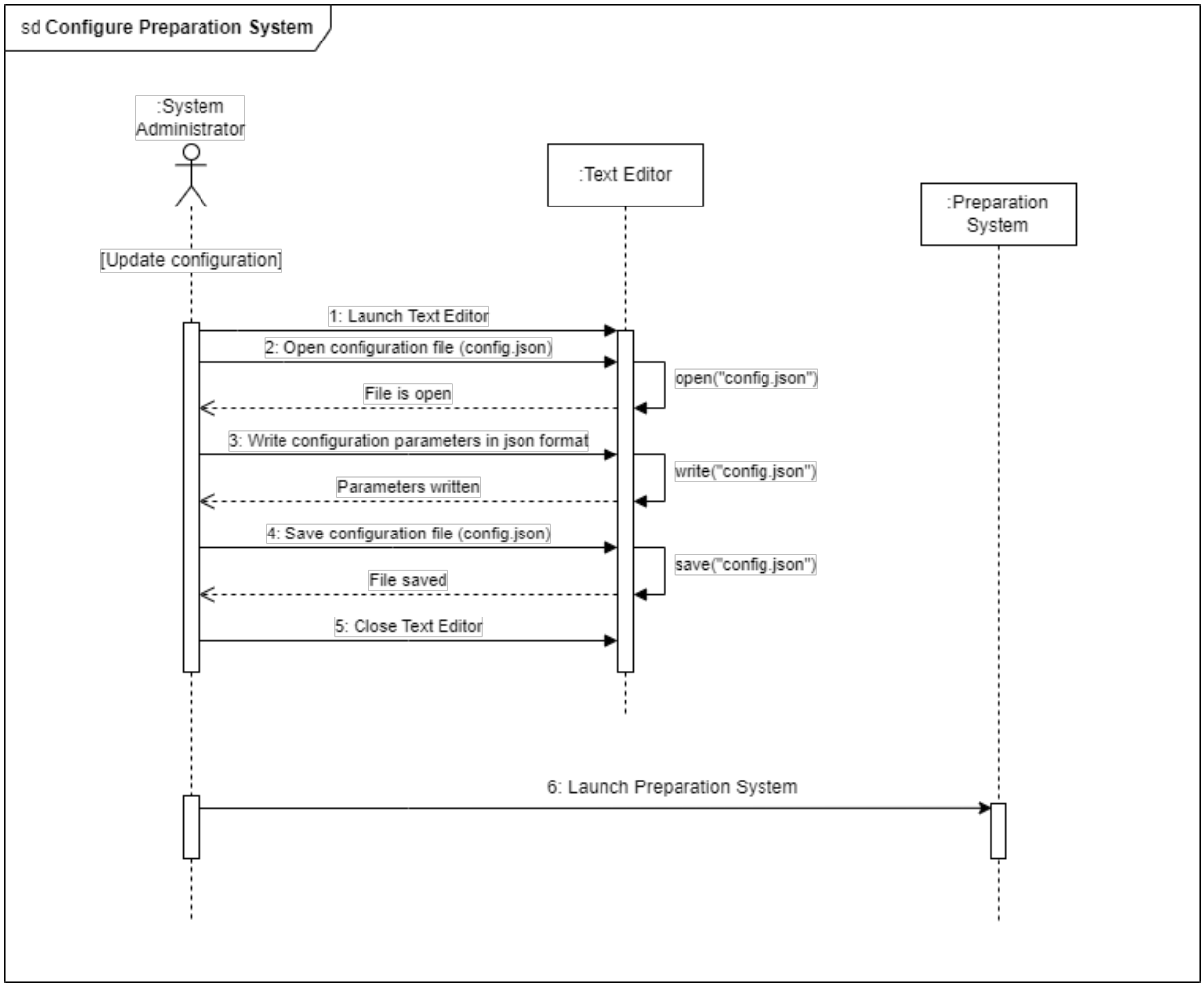

sd Configure Ingestion System
:System Administrator
:Text Editor
:Ingestion System
Launch Text Editor
Display Text Editor on screen
Open configuration file (config.json)
open("config.json")
Display file on screen
Write configuration parameters in json format
write("config.json")
Display updated file on screen
Save configuration file (config.json)
save("config.json")
File saved
Close Text Editor
Start the Ingestion System

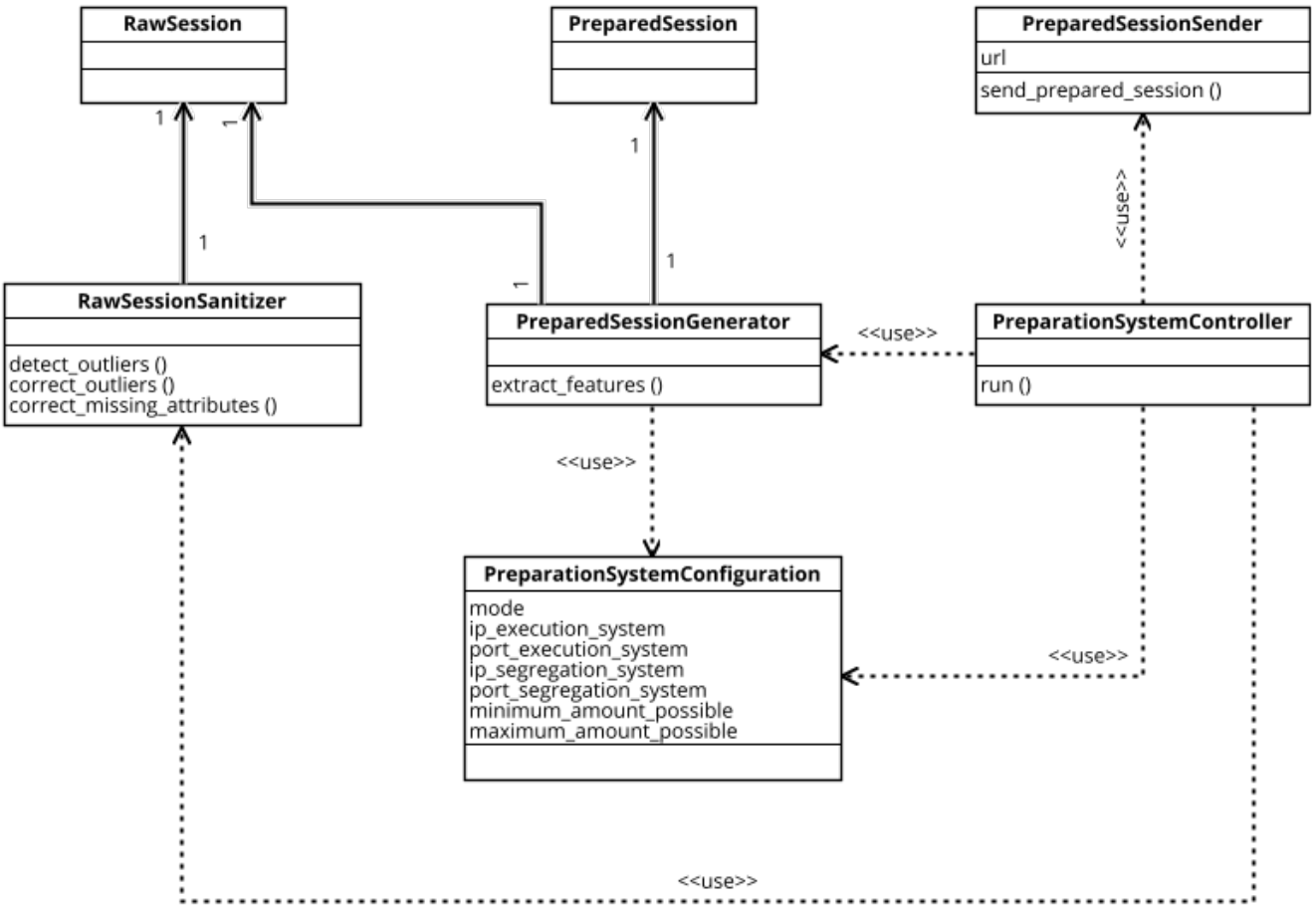

validation
GeoDataValidator
count_invalid_attributes()
NetworkDataValidator
count_invalid_attributes()
CommercialDataValidator
count_invalid_attributes()
TransactionValidator
max_invalid_attributes_allowed
is_valid()
<<use>>
data_objects
CommercialData
event_id
cardid
posid
posname
date
time
payment_type
payment_circuit
amount
currency
NetworkData
event_id
ip
GeoData
event_id
loc_name
p_id
longitude
latitude
Transaction
<<enumeration>>
AttackRiskLabel
ATTACK
NORMAL
RawSession
commercial_data
network_data
geo_data
network
commercial
geo
transactions
label
ingestion_system
AttackRiskLabelPackager
package_as_json_dict()
RawSessionPackager
package_as_json_dict()
raw_session
RecordSynchronizer
try_session_synchronization()
raw_session_synchronizer
IngestionSystemController
run()
handle_new_record_reception()
configuration
RawSessionSanitizer
remove_invalid_transactions()
Configuration
ip_address
port
database_path
preparation_system_url
monitoring_system_url
max_invalid_attributes_allowed
min_transactions_per_session
execution_window_length
monitoring_window_length
is_development_mode

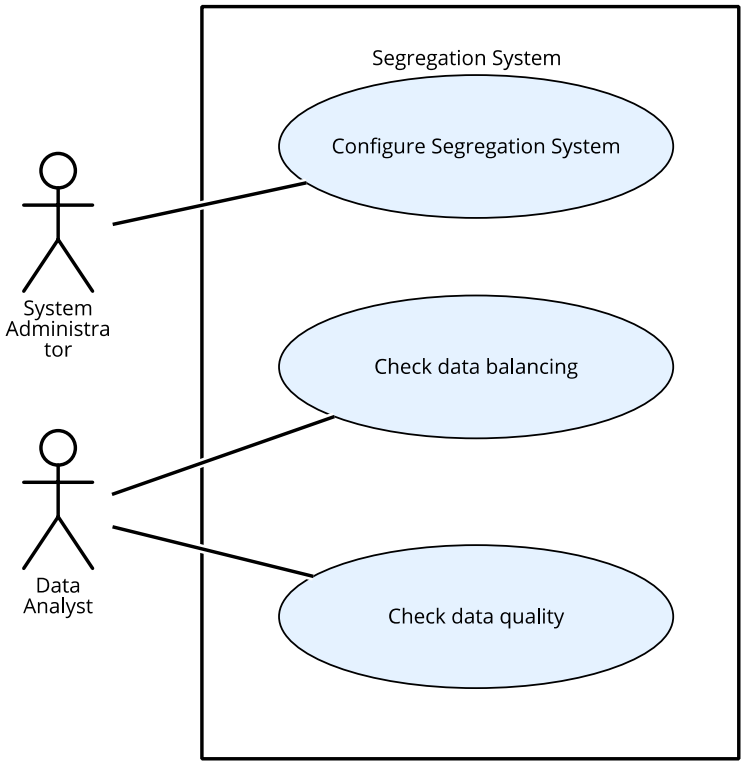
Preparation System



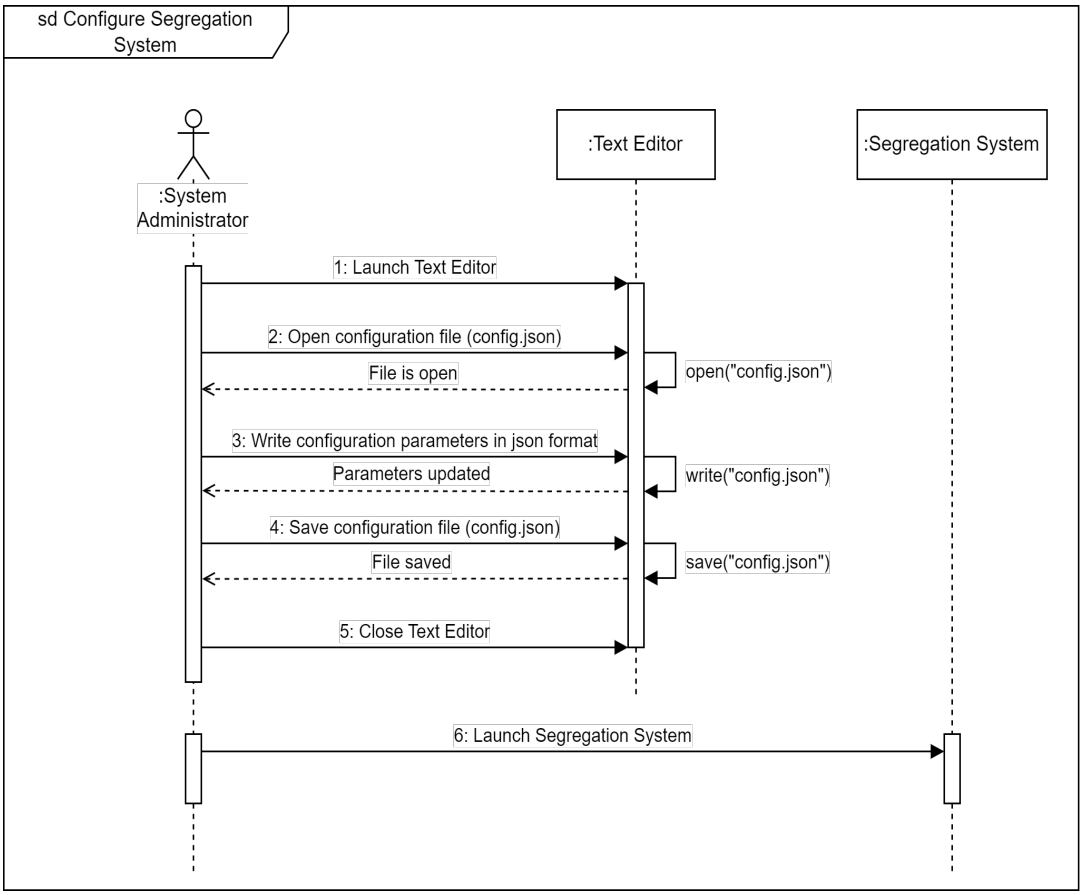
Configure Preparation System
ID: UC1
Actors: System Administrator
Preconditions: 1. The system administrator has created the file "config.json" and the validation scheme for this json file. 2. A new configuration for the preparation system is requested.
Flow of events: 1. The system administrator launches a text editor. 2. The system administrator opens the file "config.json". 3. The system administrator writes the configuration paramenters in json format, in accordance with the validation scheme for the config.json file. 4. The system administrator saves the file "config.json". 5. The system administrator closes the text editor. 6. The system administrator launches the Preparation System.
Postconditions: 1. The preparation system has been configured.


sd Configure Preparation System
:System Administrator
:Text Editor
:Preparation System
[Update configuration]
1: Launch Text Editor
2: Open configuration file (config.json)
open("config.json")
File is open
3: Write configuration parameters in json format
write("config.json")
Parameters written
4: Save configuration file (config.json)
save("config.json")
File saved
5: Close Text Editor
6: Launch Preparation System

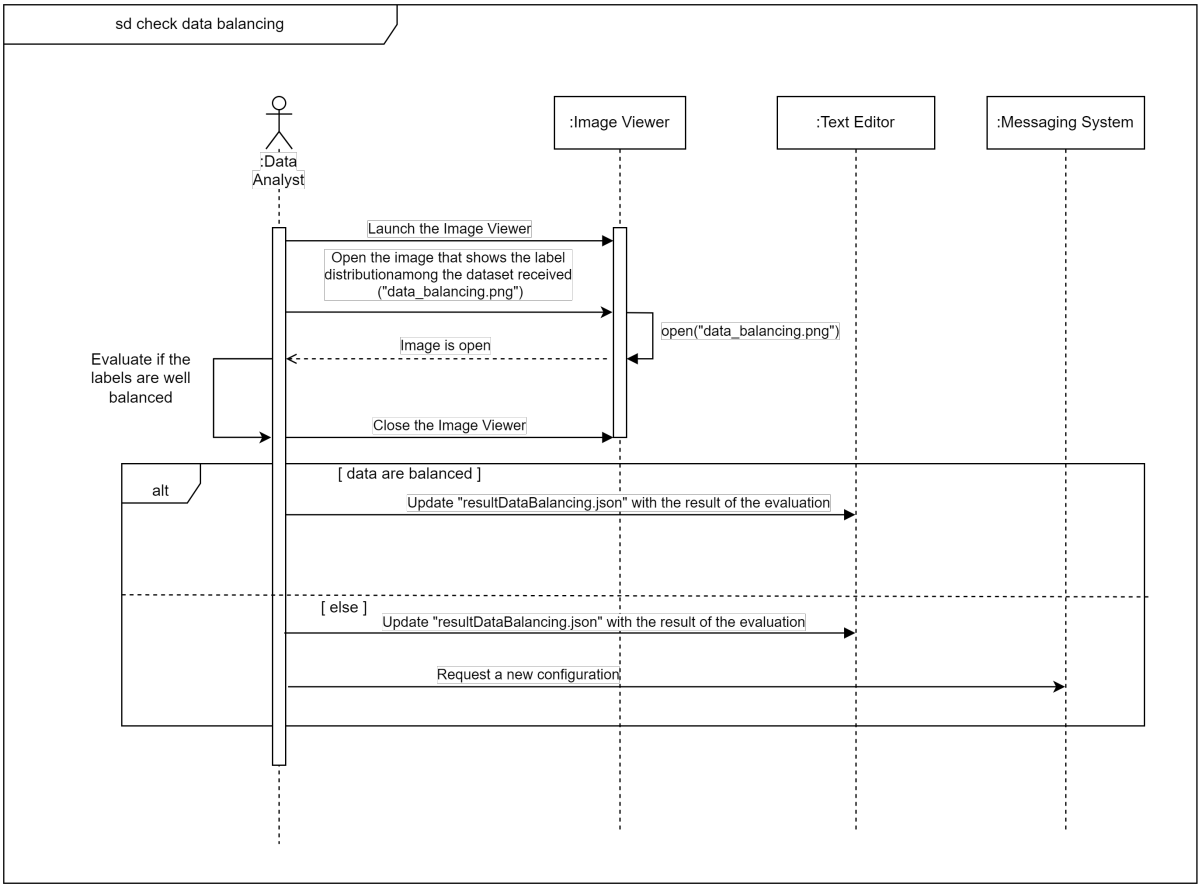

RawSession
PreparedSession
PreparedSessionSender
url
send_prepared_session ()
<<use>>
RawSessionSanitizer
detect_outliers ()
correct_outliers ()
correct_missing_attributes ()
PreparedSessionGenerator
extract_features ()
<<use>>
PreparationSystemController
run ()
<<use>>
PreparationSystemConfiguration
mode
ip_execution_system
port_execution_system
ip_segregation_system
port_segregation_system
minimum_amount_possible
maximum_amount_possible
<<use>>
<<use>>
1
1
1
1
1
1

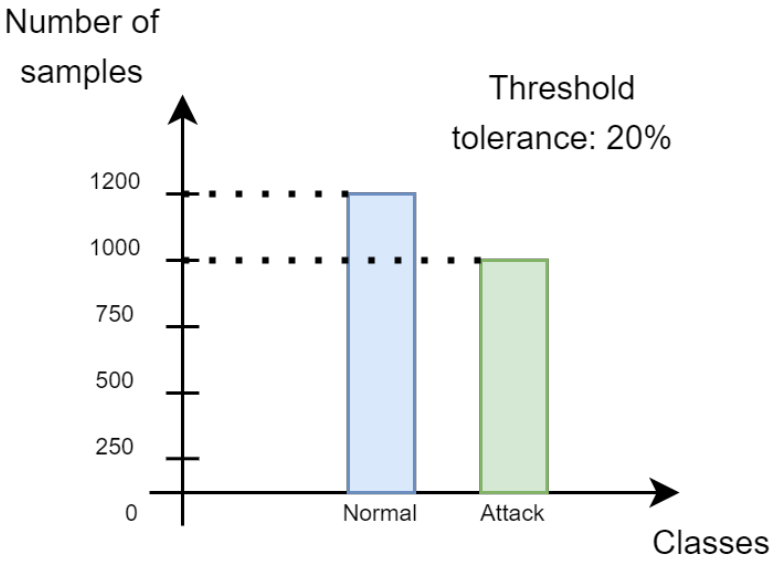


Configure Segregation System
ID: UC2
Actors: System Administrator
Preconditions: 1. A new configuration for the segregation system is requested
Flow of events: 1. The system administrator launches the Text Editor 2. The system administrator open configuration file (config.json) 3. The system administrator write configuration parameters in json format 4. The system administrator save configuration file (config.json) 5. The system administrator close the Text Editor
Postconditions: 1. The segregation system has been configured

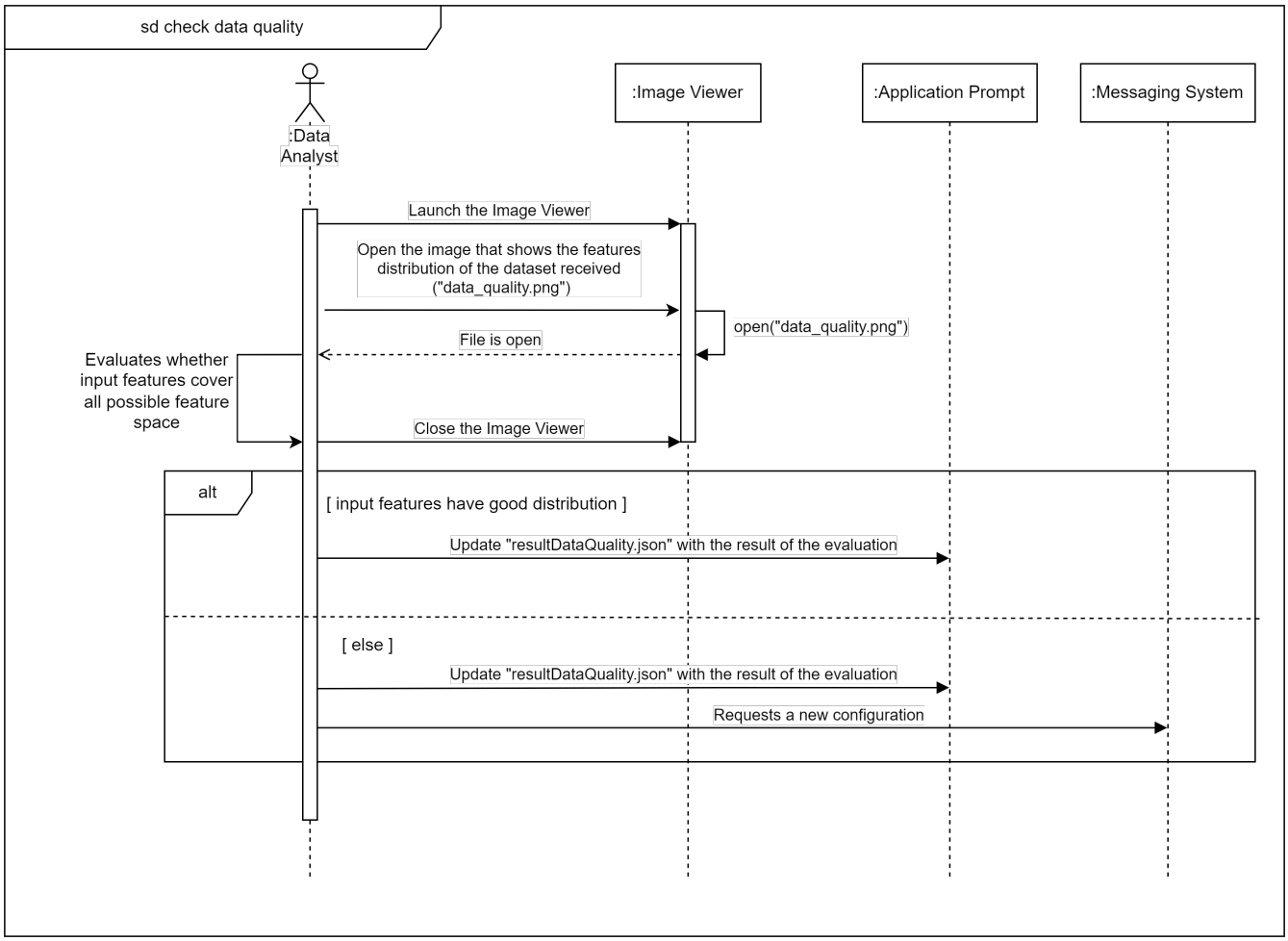

sd Configure Segregation System
:System Administrator
:Text Editor
:Segregation System
1: Launch Text Editor
2: Open configuration file (config.json)
File is open
open("config.json")
3: Write configuration parameters in json format
Parameters updated
write("config.json")
4: Save configuration file (config.json)
File saved
save("config.json")
5: Close Text Editor
6: Launch Segregation System

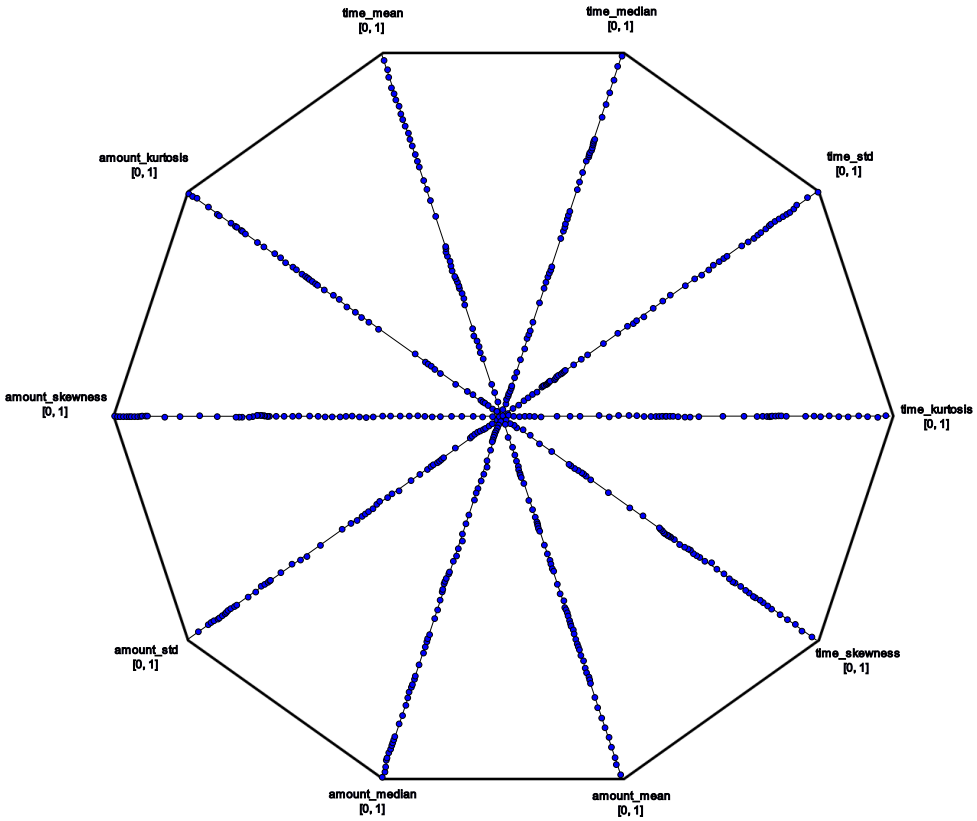
Check Data Balancing
ID: UC3
Actors: Data Analyst
Preconditions: 1. The application has produced a plot with to represent the label distribution. 2. The application has stopped its execution waiting for a user input.
Flow of events: 1. The Data Analyst launches the image viewer 2. The Data Analyst open the plot 3. The Data Analyst checks whether the data are well balanced across all labels 4. The Data Analyst closes the plot 5. If the input data are well balanced then 5.2 The Data Analyst enters “yes” in a JSON file 6. Else 6.2 The Data Analyst enter “no” in a JSON file 6.3 The Data Analyst request a new configuration
Postconditions:
Alternative flow 1: 1. The system continue its execution
Postconditions:
Alternative flow 2: 1. A new configuration is requested

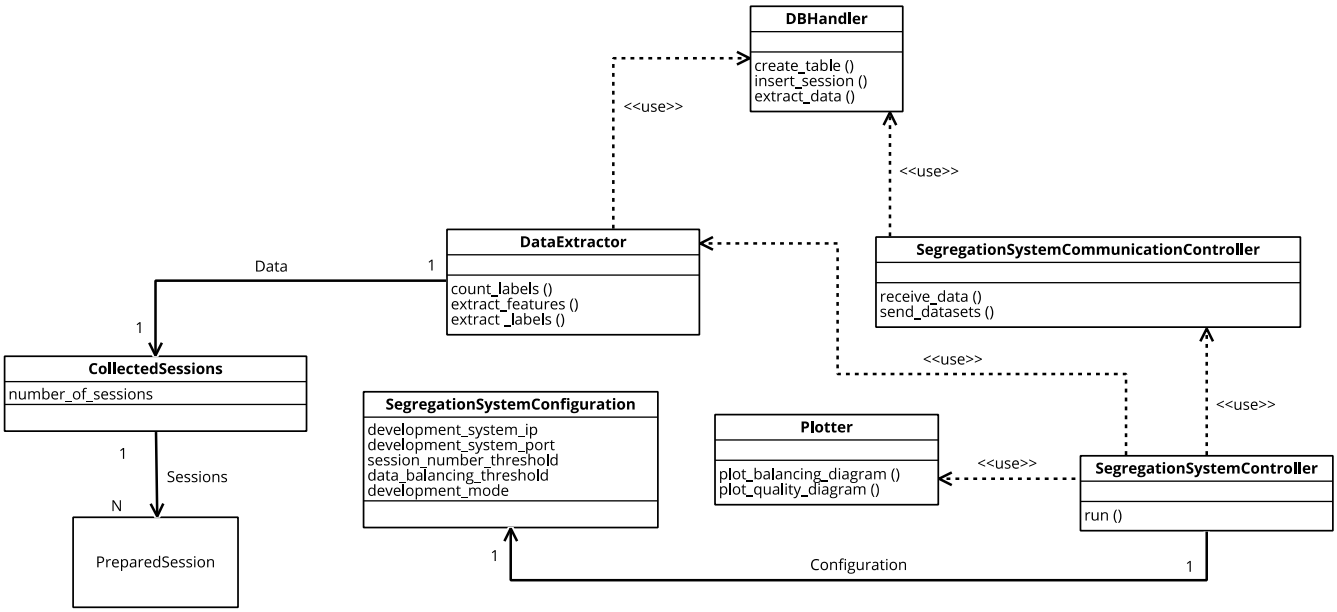




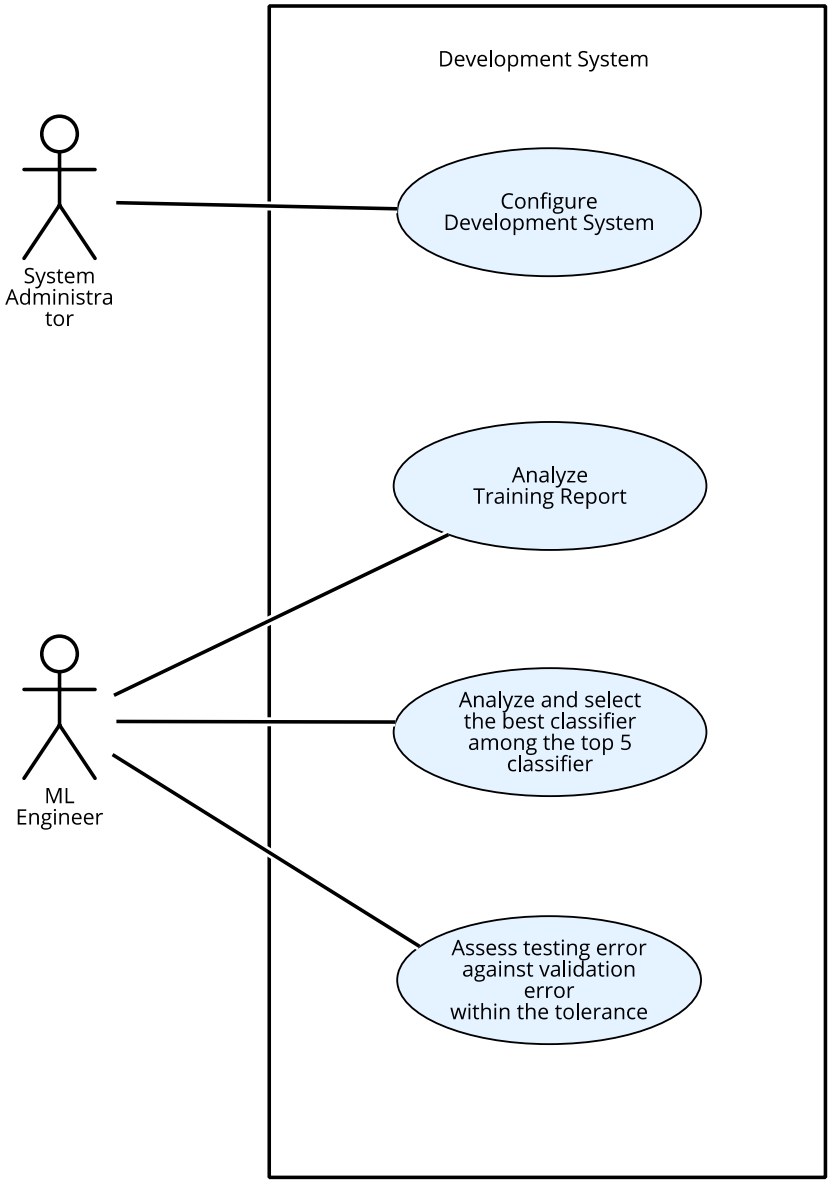
Check Data Quality
ID: UC4
Actors: Data Analyst
Preconditions: 1. The application has produced plot with to assess the data quality 2. The application has stopped its execution waiting for a user input.
Flow of events: 1. The Data Analyst launches the image viewer 2. The Data Analyst open the plot 3. The Data Analyst observes whether the data are well balanced based on the diagram 4. The Data Analyst closes the file 5. If data have a good distribution then 4.1 The Data Analyst enters “yes” in a JSON file 6. Else 5.1 The Data Analyst enters “no” in a JSON file 5.2 The Data Analysts requests a new configuration
Postconditions:
Alternative flow 1: 1. The datasets are generated
Postconditions:
Alternative flow 2: 1. A new configuration is requested


sd check data quality
:Data Analyst
:Image Viewer
:Application Prompt
:Messaging System
Launch the Image Viewer
Open the image that shows the features distribution of the dataset received ("data_quality.png")
open("data_quality.png")
File is open
Evaluates whether input features cover all possible feature space
Close the Image Viewer
alt
[input features have good distribution]
Update "resultDataQuality.json" with the result of the evaluation
[else]
Update "resultDataQuality.json" with the result of the evaluation
Requests a new configuration

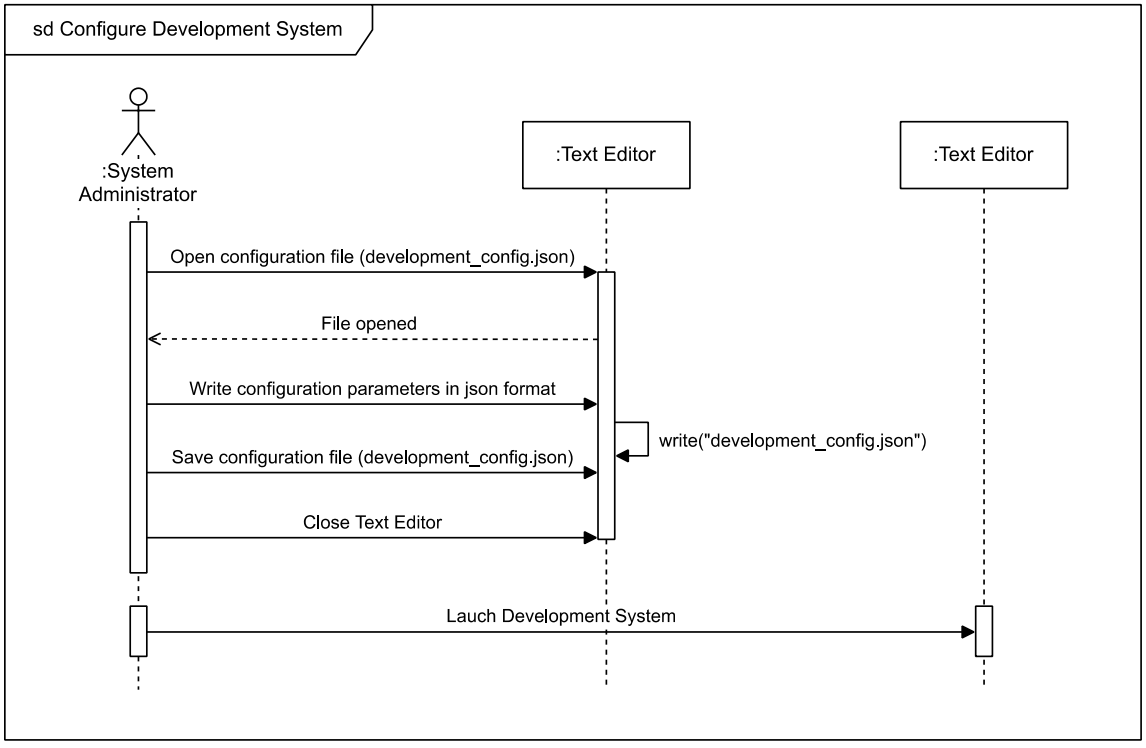

time_mean [0, 1]
time_median [0, 1]
time_std [0, 1]
time_kurtosis [0, 1]
time_skewness [0, 1]
amount_mean [0, 1]
amount_median [0, 1]
amount_std [0, 1]
amount_skewness [0, 1]
amount_kurtosis [0, 1]


DBHandler
create_table ()
insert_session ()
extract_data ()
<<use>>
<<use>>
DataExtractor
count_labels ()
extract_features ()
extract_labels ()
SegregationSystemCommunicationController
receive_data ()
send_datasets ()
Data
1
1
CollectedSessions
number_of_sessions
1
Sessions
N
PreparedSession
SegregationSystemConfiguration
development_system_ip
development_system_port
session_number_threshold
data_balancing_threshold
development_mode
Plotter
plot_balancing_diagram ()
plot_quality_diagram ()
<<use>>
<<use>>
<<use>>
SegregationSystemController
run ()
1
Configuration
1

Development System



Configure Development System
ID: UC5
Actors: System Administrator
Preconditions: 1. The system administrator has created the file "development_config.json". 2. A new configuration for the development system is requested
Flow of events: 1. The system administrator launches the Text Editor 2. The system administrator opens the configuration file (development_config.json) 3. The system administrator writes configuration parameters in json format 4. The system administrator saves the configuration file (development_config.json) 5. The system administrator closes the Text Editor
Postconditions: 1. The development system has been configured



Analyze Training Report

ID: UC6

Actors: Machine Learning Engineer

Preconditions:

1. The ML Engineer has created the file "*training_configuration.json*".
2. The application has produced a Training Report, which consists of a json file containing the results of the training ("*report_initial_phase_training.json*") and the plot of the loss function.
3. The application has stopped its execution to wait for the actor's analysis and input.

Flow of events:

1. The ML Engineer opens the plot of the loss function.
2. The ML Engineer opens the *" report_initial_phase_training.json "* file.
3. The ML Engineer analyze each component of the Training Report.
4. The ML Engineer opens the "*training_configuration.json*" file.
5. If the number of generations is correct, then:
   - **5.1.** The ML Engineer sets the **is_initial_phase_over parameter** to "Yes" in the *"training_configuration.json "* file.
6. Else:
   - **6.1.** If the loss function plot is flat for at least half of the generations, then:
     - 6.1.1. The ML Engineer updates the "*training_configuration.json*" file by reducing the number of generations, in order to manage overfitting.
   - **6.2.** Else:
     - 6.2.1. The ML Engineer updates the "*training_configuration.json*" file by increasing the number of generations.
7. The ML Engineer saves the *" training_configuration.json "* file.
8. The ML Engineer closes the Text Editor.
9. The ML Engineer closes the Image Viewer.

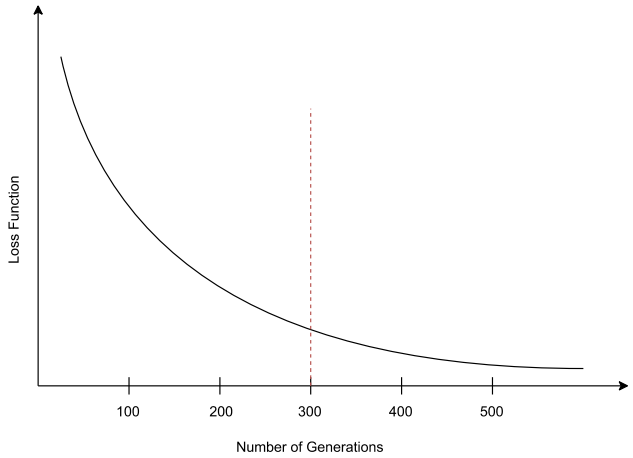
Postconditions:

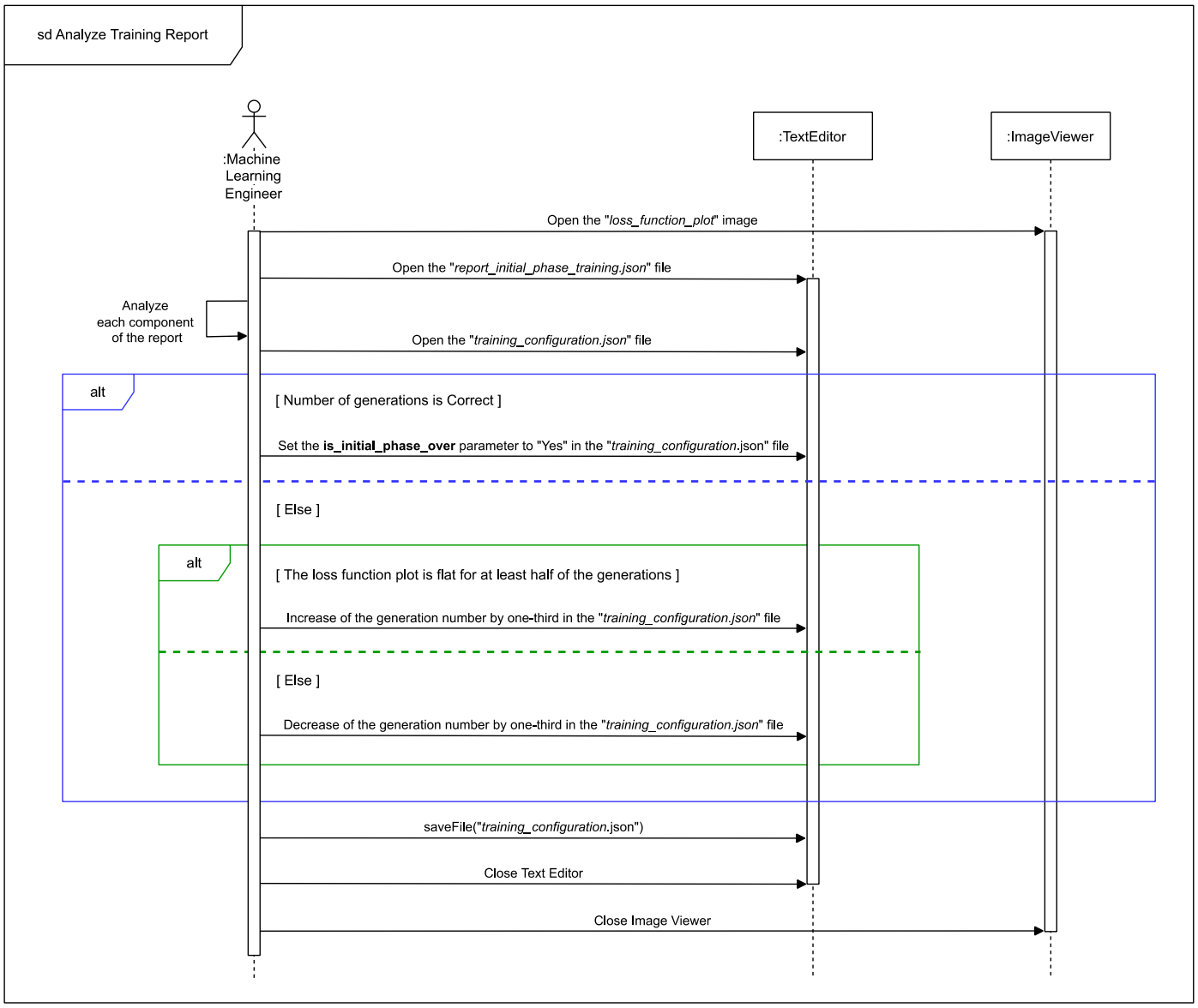
- Alternative Flow 1:

1. The Training Report has been analyzed.
2. The system continued with its execution by setting the hyperparameters according to the grid search

- Alternative Flow 2:

1. The number of generations has been tuned.
2. A new neural network training has started.


Loss Function
100
200
300
400
500
Number of Generations


sd Analyze Training Report
:Machine Learning Engineer
:TextEditor
:ImageViewer
Open the "loss_function_plot" image
Open the "report_initial_phase_training.json" file
Analyze each component of the report
Open the "training_configuration.json" file
alt
[Number of generations is Correct]
Set the is_initial_phase_over parameter to "Yes" in the "training_configuration.json" file
[Else]
alt
[The loss function plot is flat for at least half of the generations]
Increase of the generation number by one-third in the "training_configuration.json" file
[Else]
Decrease of the generation number by one-third in the "training_configuration.json" file
saveFile("training_configuration.json")
Close Text Editor
Close Image Viewer

Analyze and Select the best classifier among the top 5 classifiers

ID: UC7

Actors:
ML Engineer

Preconditions:
1. The report of the top 5 classifiers has been generated.

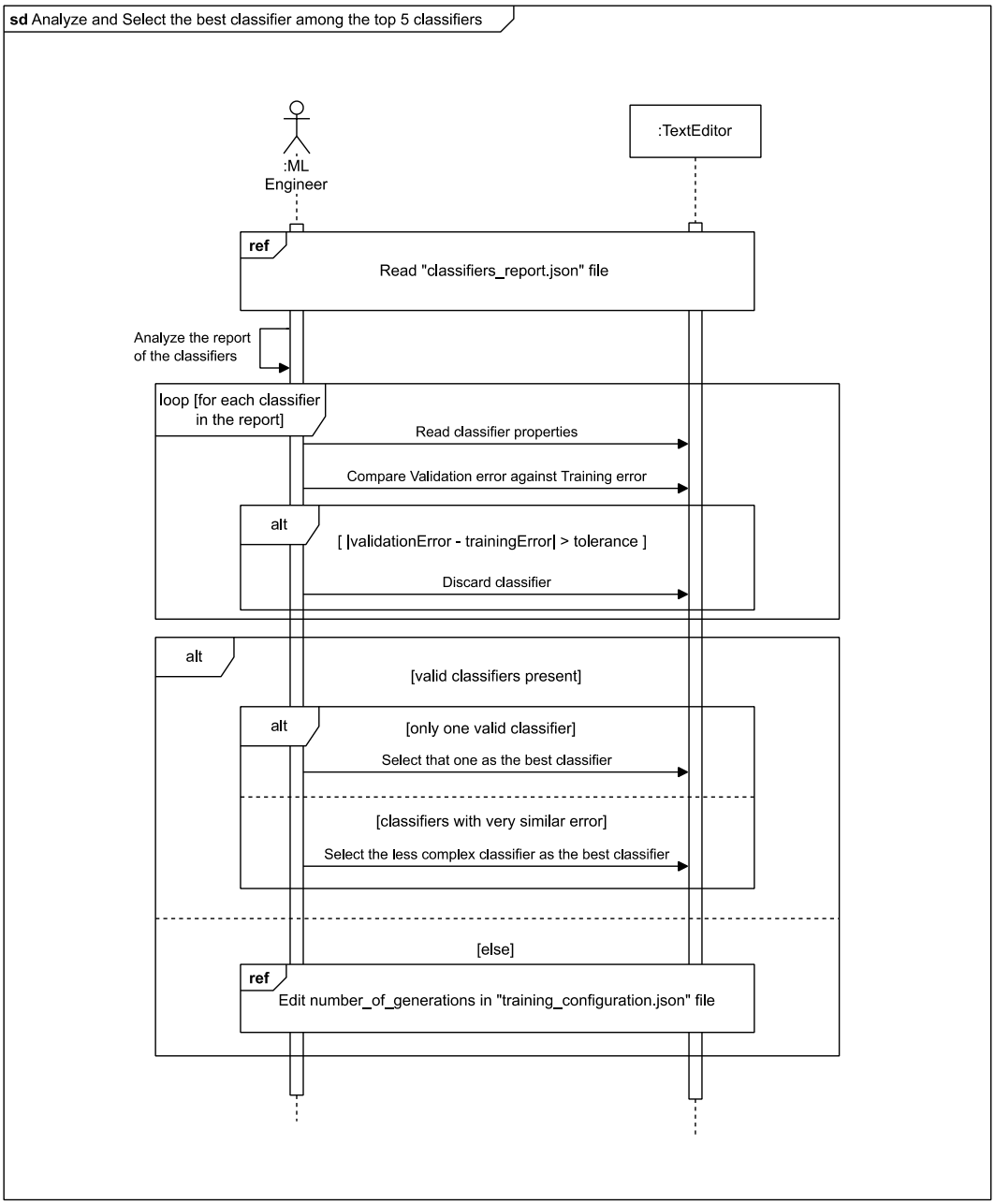
1. **Flow of events:**
   2. The ML Engineer opens the file containing the report of the classifiers..
   3. The ML Engineer analyzes the report.
   4. For each classifier in the report
      4.1. The ML Engineer compares the validation error against the training error.
      4.2. If the difference is greater than the given tolerance threshold
         4.2.1. The classifier is discarded.
   5. If there are any valid classifiers left
      5.1. If there is only one valid classifier left
         5.1.1. The ML Engineer chooses that classifier as the best.
      5.2. Else if there are more than one classifier with very similar error
         5.2.1. The ML Engineer chooses the less complex one.
   6. Else
      6.1. The ML Engineer opens the "training_configuration.json" file.
      6.2. The ML Engineer reduces the number of generations to manage overfitting.
      6.3. The ML Engineer updates the "training_configuration.json" file.
   7. The ML Engineer saves the "training_configuration.json" file.

Postconditions:

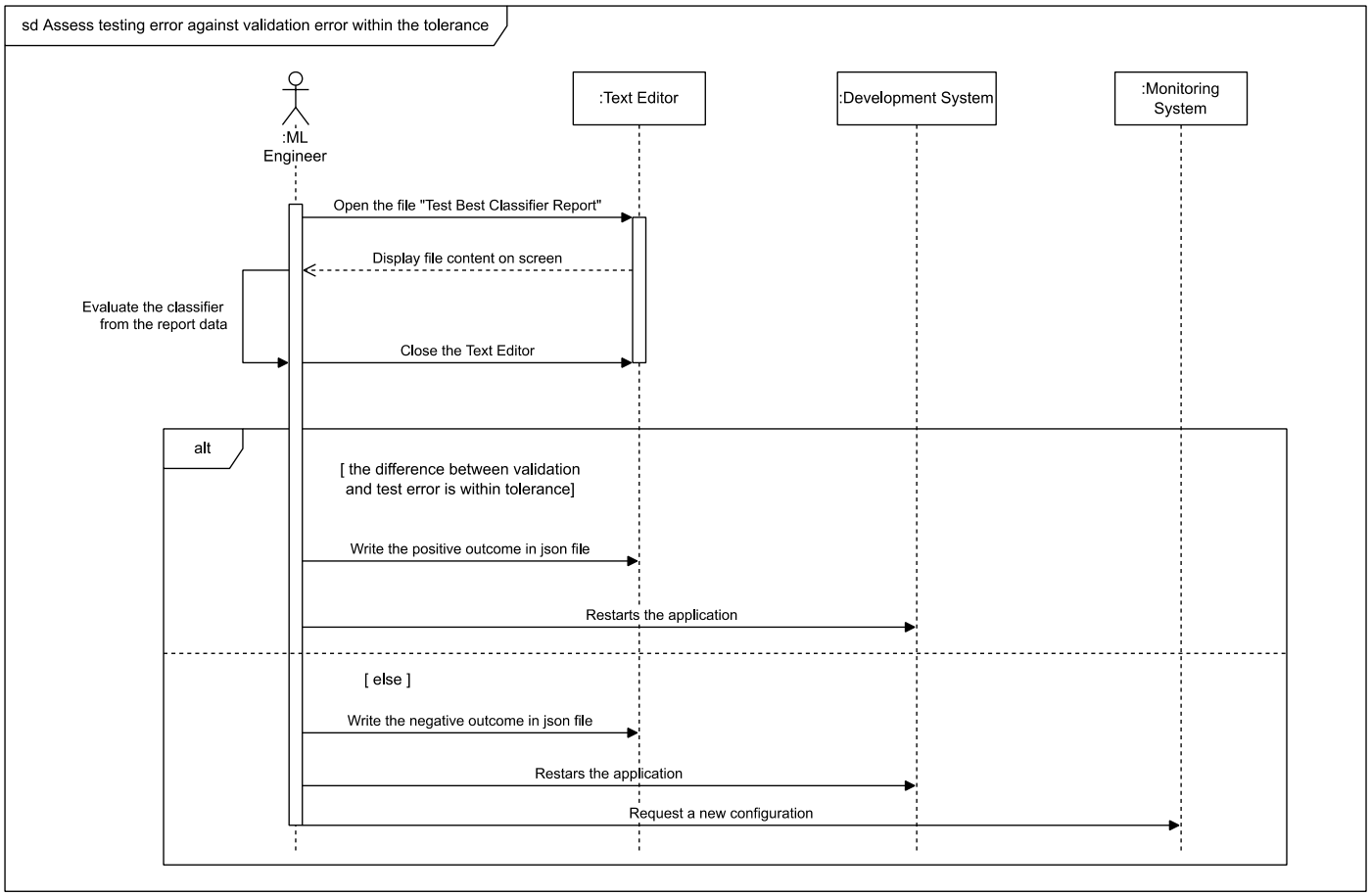
Alternative flow 1:
1. The best classifier has been selected.

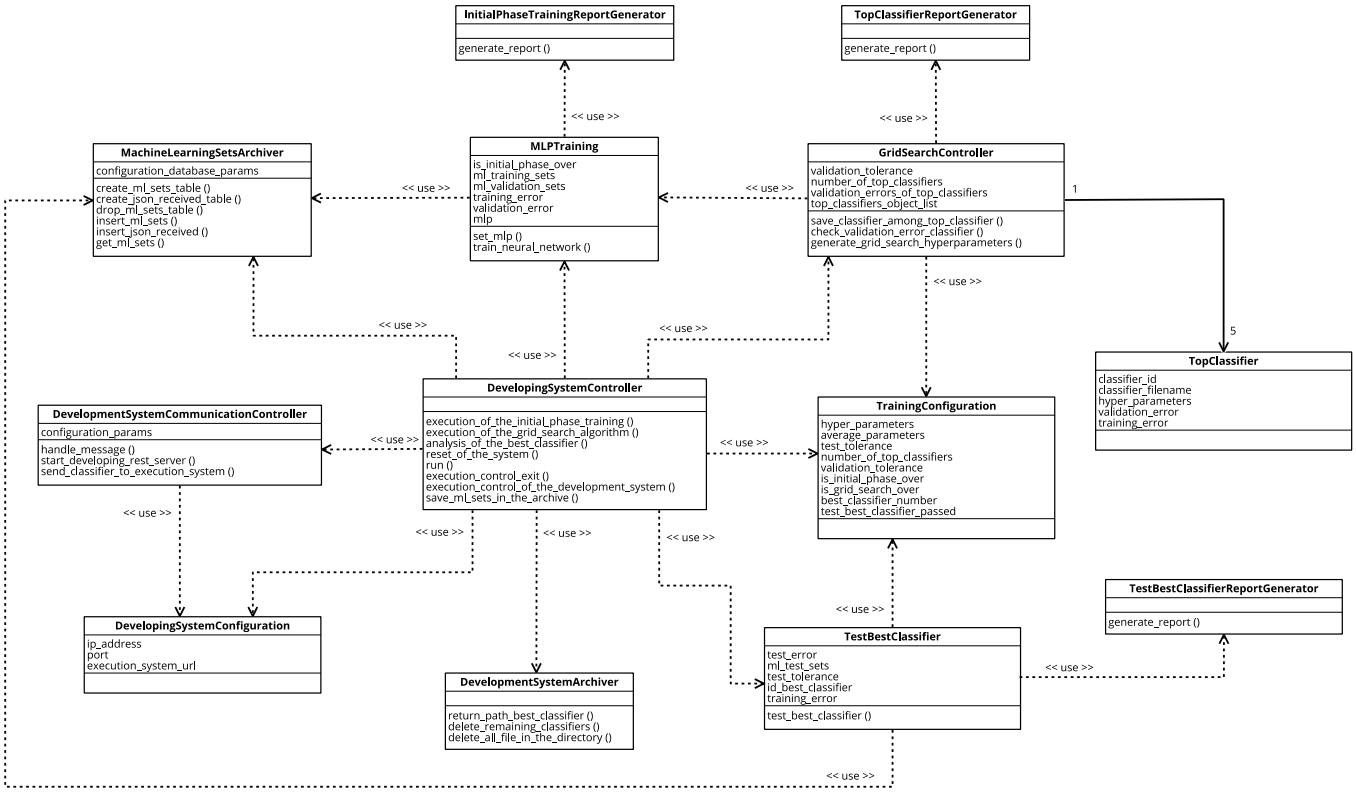
Postconditions:

Alternative flow 2:
1. The number of generations has been tuned.
2. A new training has started.

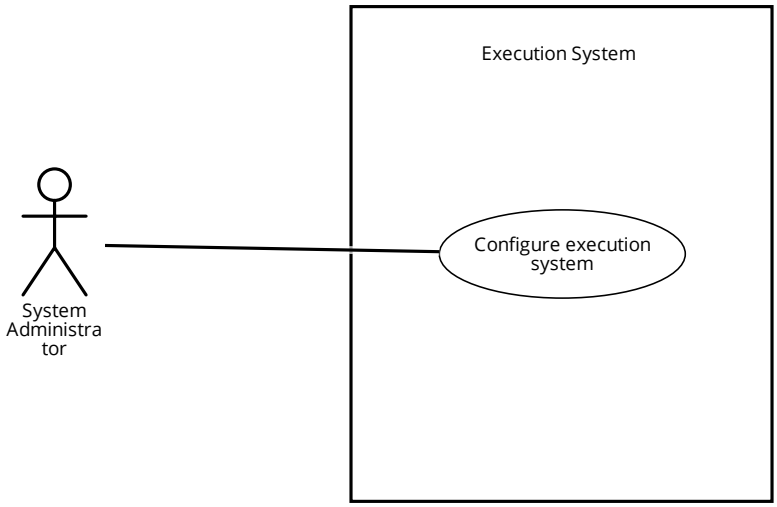

sd Analyze and Select the best classifier among the top 5 classifiers
:ML Engineer
:TextEditor
ref
Read "classifiers_report.json" file
Analyze the report of the classifiers
loop [for each classifier in the report]
Read classifier properties
Compare Validation error against Training error
alt
[|validationError - trainingError| > tolerance]
Discard classifier
alt
[valid classifiers present]
alt
[only one valid classifier]
Select that one as the best classifier
[classifiers with very similar error]
Select the less complex classifier as the best classifier
[else]
ref
Edit number_of_generations in "training_configuration.json" file

Assess testing error against validation error within the tolerance
ID: UC8
Actors: ML Engineer
Preconditions: 1. The application of the Development System has tested the best classifier. 2. The application has produced a report file with the results of the test. 3. The application has terminated its execution.
Flow of events: 1. The ML Engineer launches the Text Editor. 2. The ML Engineer opens the report file (“Test Best Classifier Report”). 3. The ML Engineer evaluates the classifier from the report data. 4. The ML Engineer closes the Text Editor. 5. If the difference between validation and test error is within the tolerance then 5.1. The ML Engineer notifies the positive outcome in a json file. 5.2. The ML Engineer restarts the application of the Development System. 6. Else 6.1. The ML Engineer notifies the negative outcome in a json file. 6.2. The ML Engineer restarts the application of the Development System, in order to clear the system. 6.3. The ML Engineer requests a new configuration.
Postconditions:
Alternative flow 1: 1. The classifier is deployed.
Postconditions:
Alternative flow 2: 1. The classifier is rejected. 2. A new configuration has been requested

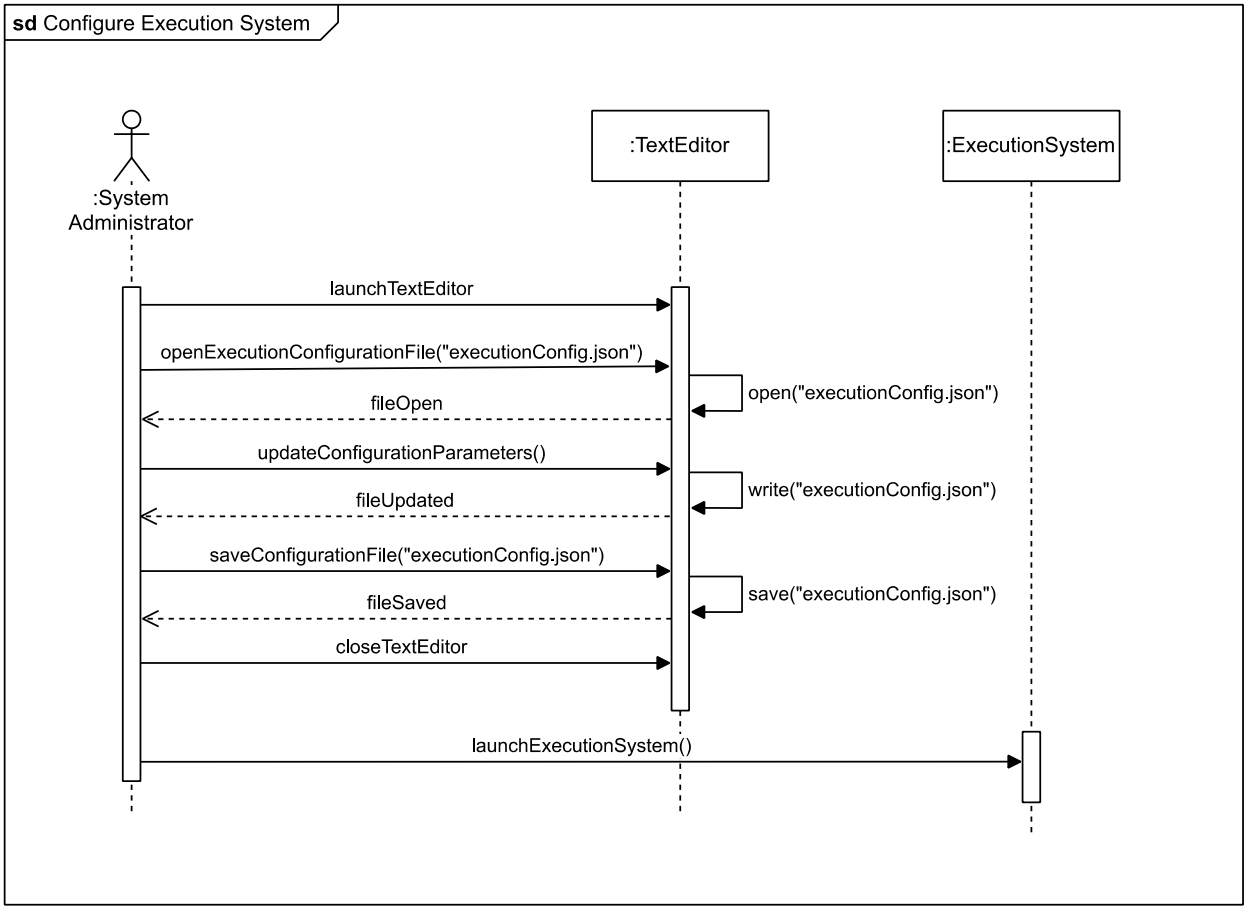

sd Assess testing error against validation error within the tolerance
:ML Engineer
:Text Editor
:Development System
:Monitoring System
Open the file "Test Best Classifier Report"
Display file content on screen
Evaluate the classifier from the report data
Close the Text Editor
alt
[the difference between validation and test error is within tolerance]
Write the positive outcome in json file
Restarts the application
[else]
Write the negative outcome in json file
Restars the application
Request a new configuration

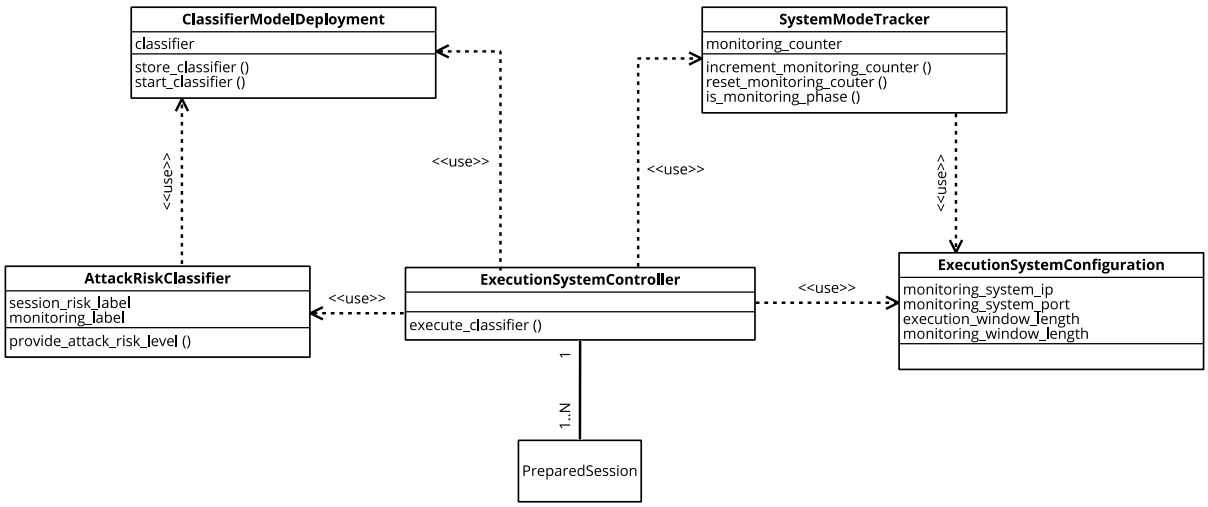

InitialPhaseTrainingReportGenerator
generate_report ()
TopClassifierReportGenerator
generate_report ()
<< use >>
MachineLearningSetsArchiver
configuration_database_params
create_ml_sets_table ()
create_json_received_table ()
drop_ml_sets_table ()
insert_ml_sets ()
insert_json_received ()
get_ml_sets ()
MLPTraining
is_initial_phase_over
ml_training_sets
ml_validation_sets
training_error
validation_error
mlp
set_mlp ()
train_neural_network ()
GridSearchController
validation_tolerance
number_of_top_classifiers
validation_errors_of_top_classifiers
top_classifiers_object_list
save_classifier_among_top_classifier ()
check_validation_error_classifier ()
generate_grid_search_hyperparameters ()
1
5
TopClassifier
classifier_id
classifier_filename
hyper_parameters
validation_error
training_error
DevelopingSystemController
execution_of_the_initial_phase_training ()
execution_of_the_grid_search_algorithm ()
analysis_of_the_best_classifier ()
reset_of_the_system ()
run ()
execution_control_exit ()
execution_control_of_the_development_system ()
save_ml_sets_in_the_archive ()
DevelopmentSystemCommunicationController
configuration_params
handle_message ()
start_developing_rest_server ()
send_classifier_to_execution_system ()
TrainingConfiguration
hyper_parameters
average_parameters
test_tolerance
number_of_top_classifiers
validation_tolerance
is_initial_phase_over
is_grid_search_over
best_classifier_number
test_best_classifier_passed
DevelopingSystemConfiguration
ip_address
port
execution_system_url
DevelopmentSystemArchiver
return_path_best_classifier ()
delete_remaining_classifiers ()
delete_all_file_in_the_directory ()
TestBestClassifier
test_error
ml_test_sets
test_tolerance
id_best_classifier
training_error
test_best_classifier ()
TestBestClassifierReportGenerator
generate_report ()

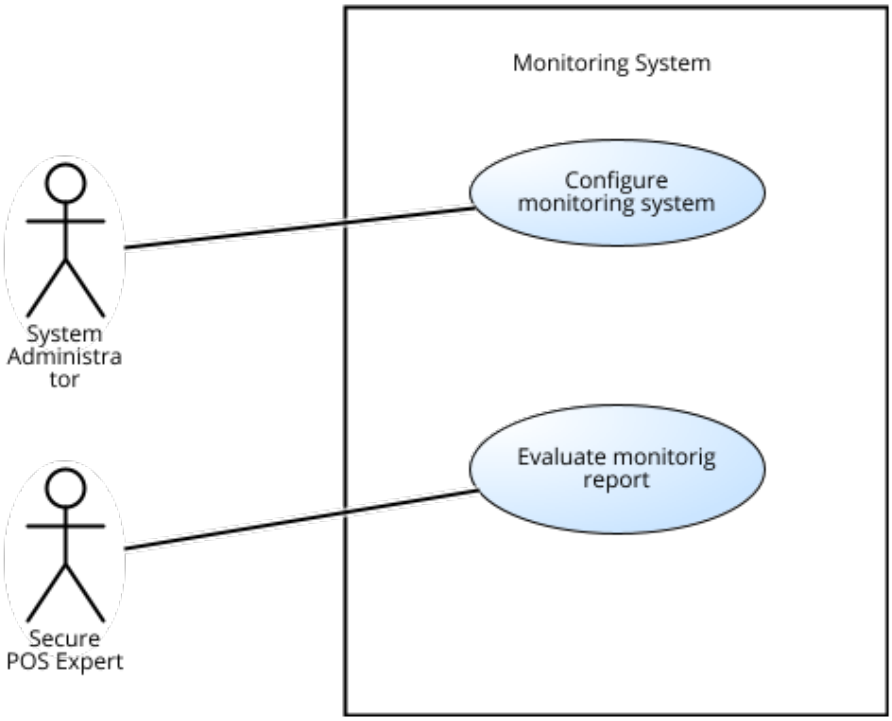
Execution System



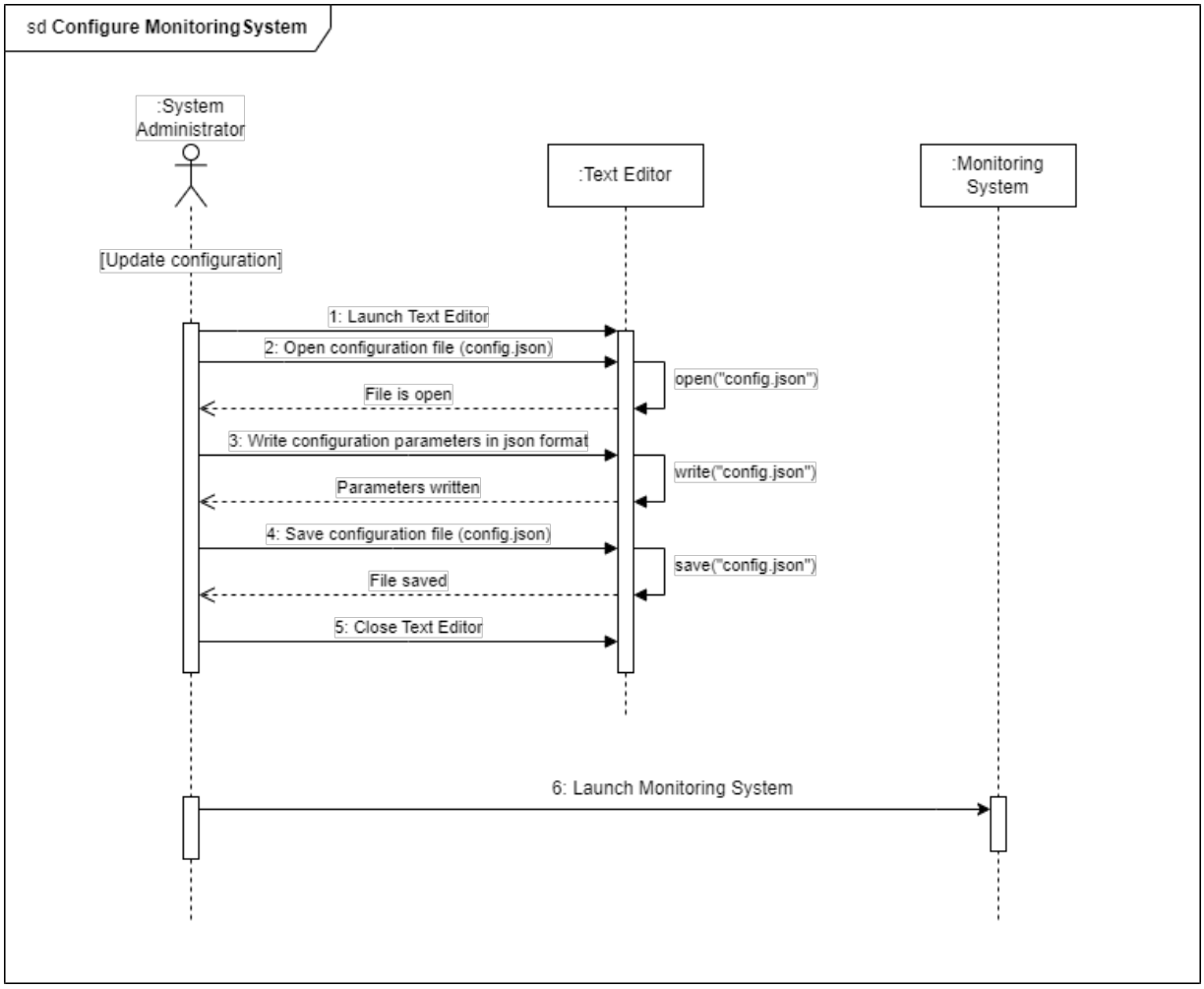
Configure Execution System
ID: UC9
Actors: System Administrator
Preconditions: 1. A new configuration is requested from the Execution System.
Flow of events: 1. The System Administrator launches a Text Editor. 2. The System Administrator opens the "executionConfig.json" file. 3. The System Administrator updates the "executionConfig.json" file with the new configuration parameters. 4. The System Administrator saves the "executionConfig.json" file. 5. The System Administrator closes the Text Editor.
Postconditions: 1. The Execution System has been configured.


sd Configure Execution System
:System Administrator
:TextEditor
:ExecutionSystem
launchTextEditor
openExecutionConfigurationFile("executionConfig.json")
open("executionConfig.json")
fileOpen
updateConfigurationParameters()
write("executionConfig.json")
fileUpdated
saveConfigurationFile("executionConfig.json")
save("executionConfig.json")
fileSaved
closeTextEditor
launchExecutionSystem()

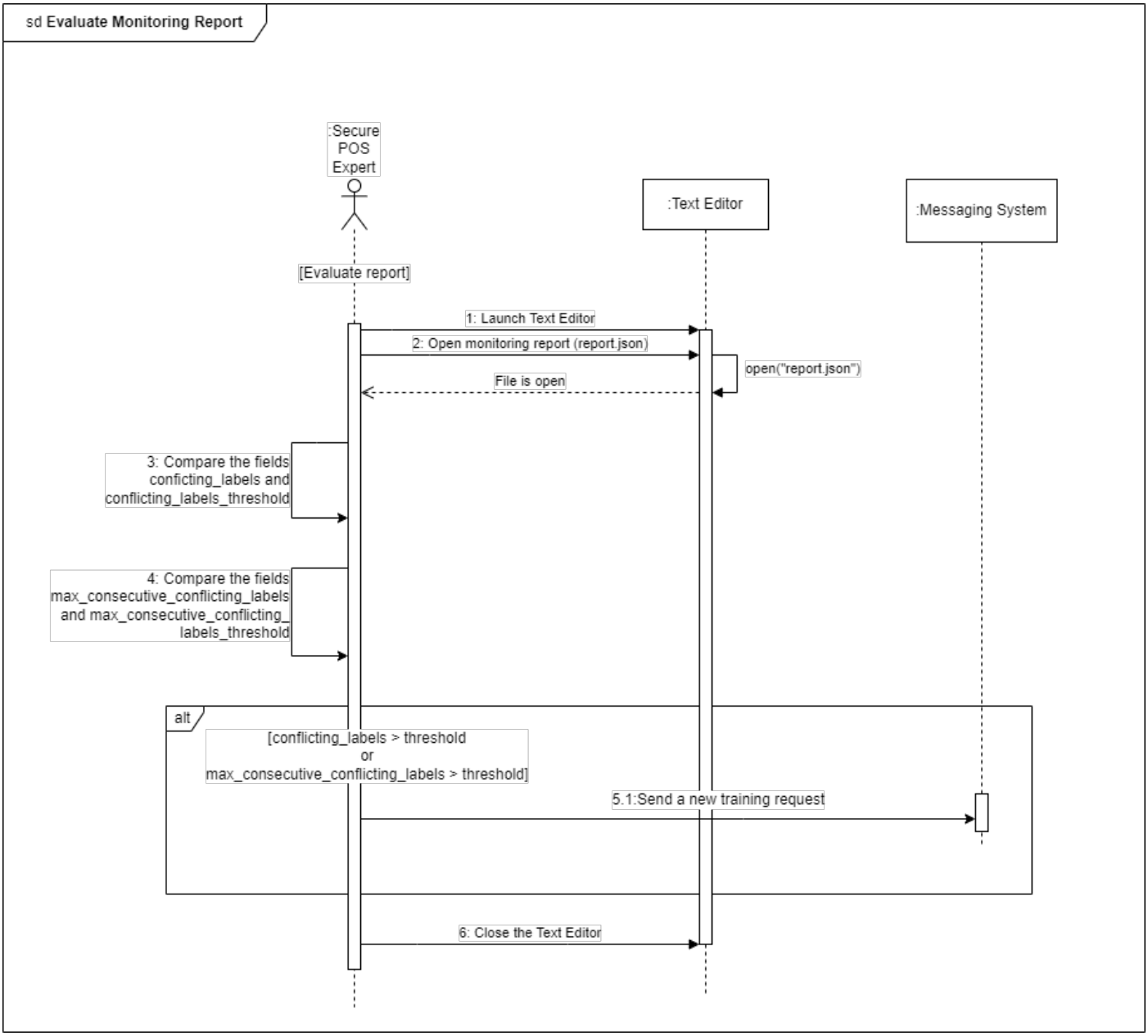

ClassifierModelDeployment
classifier
store_classifier ()
start_classifier ()
SystemModeTracker
monitoring_counter
increment_monitoring_counter ()
reset_monitoring_couter ()
is_monitoring_phase ()
<<use>>
<<use>>
<<use>>
<<use>>
AttackRiskClassifier
session_risk_label
monitoring_label
provide_attack_risk_level ()
<<use>>
ExecutionSystemController
execute_classifier ()
<<use>>
ExecutionSystemConfiguration
monitoring_system_ip
monitoring_system_port
execution_window_length
monitoring_window_length
1
1..N
PreparedSession

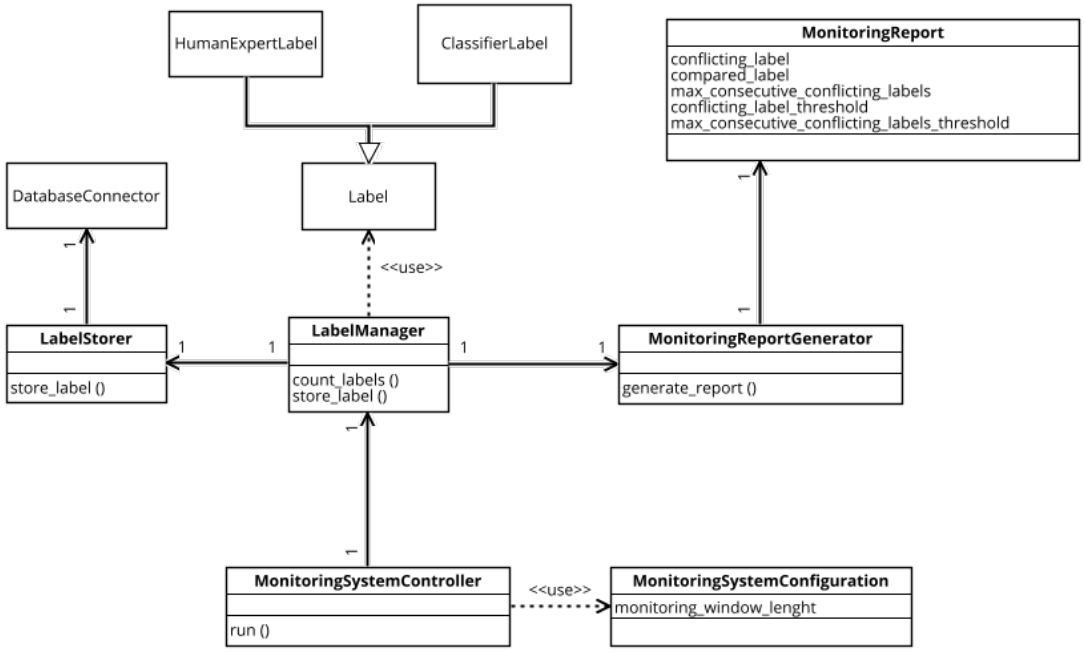


Configure Monitoring System
ID: UC11
Actors: System Administrator
Preconditions: 1. The system administrator has created the file "config.json" and the validation scheme for this json file. 2. A new configuration for the monitoring system is requested.
Flow of events: 1. The system administrator launches a text editor. 2. The system administrator opens the file "config.json". 3. The system administrator writes the configuration parameters in json format, in accordance with the validation schema. 4. The system administrator saves the file "config.json". 5. The system administrator closes the text editor. 6. The system administrator launches the Monitoring System.
Postconditions: 1. The monitoring system has been configured.


sd Configure Monitoring System
:System Administrator
:Text Editor
:Monitoring System
[Update configuration]
1: Launch Text Editor
2: Open configuration file (config.json)
open("config.json")
File is open
3: Write configuration parameters in json format
write("config.json")
Parameters written
4: Save configuration file (config.json)
save("config.json")
File saved
5: Close Text Editor
6: Launch Monitoring System

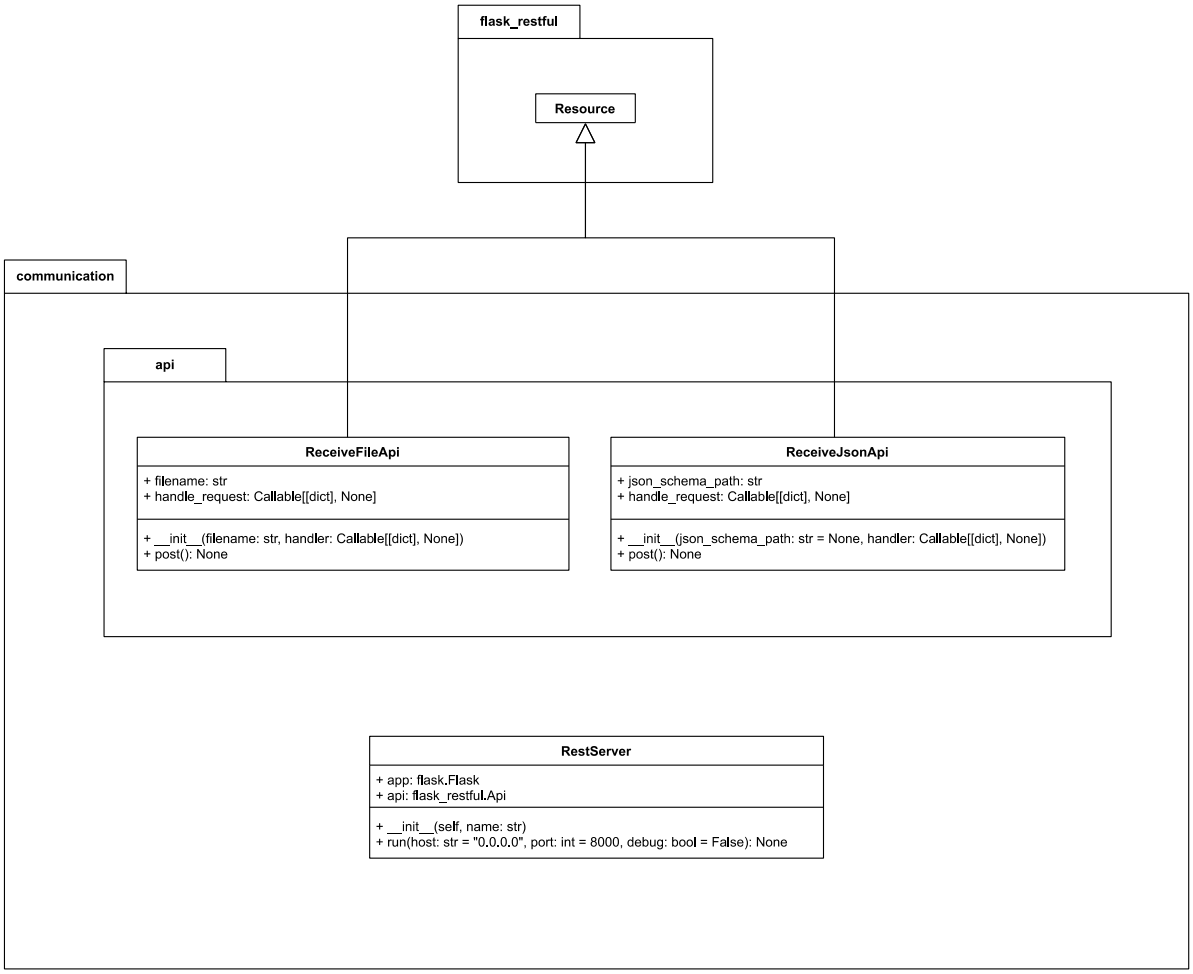
Evaluate monitorig report
ID: UC11
Actors: Secure POS Expert
Preconditions: 1. The monitoring system has generated the monitoring report, and saved it in the "report.json" file.
Flow of events: 1. The secure POS expert launches a text editor. 2. The secure POS expert opens the file "report.json". 3.1 The secure POS expert compares the two fileds conflicting labels and conflicting labels threshold. 3. The secure POS expert compares the two fileds max consecutive conflicting labels and max consecutive conflicting labels threshold. 4. If conflicting labels > threshold or max consecutive conflicting labels > threshold: 3.1.1 The secure POS expert sends a new training request, so a request for a new configuration. 3.2 The secure POS expert closes the text editor.
Postconditions: 1. The monitoring report has been evaluate.

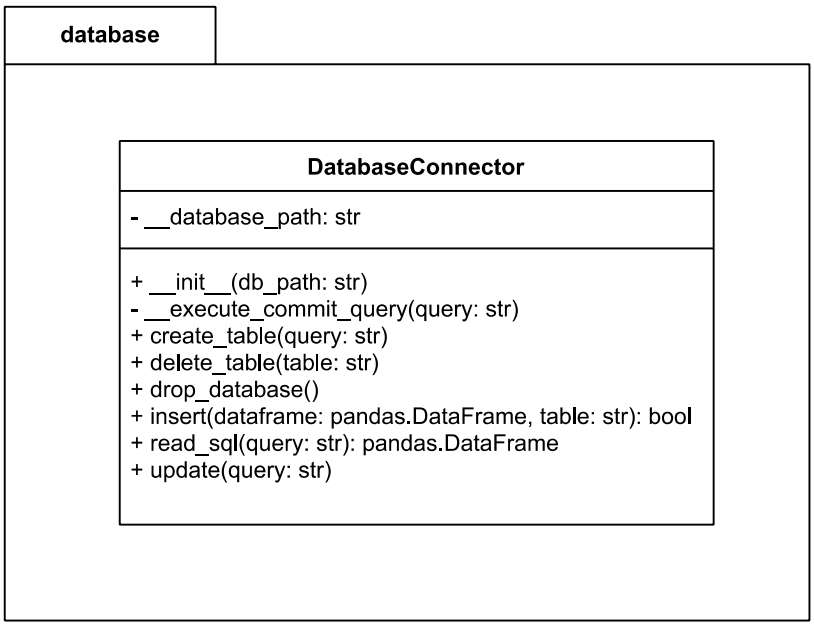

sd Evaluate Monitoring Report
:Secure POS Expert
:Text Editor
:Messaging System
[Evaluate report]
1: Launch Text Editor
2: Open monitoring report (report.json)
open("report.json")
File is open
3: Compare the fields conficting_labels and conflicting_labels_threshold
4: Compare the fields max_consecutive_conflicting_labels and max_consecutive_conflicting_labels_threshold
alt
[conflicting_labels > threshold or max_consecutive_conflicting_labels > threshold]
5.1:Send a new training request
6: Close the Text Editor


HumanExpertLabel
ClassifierLabel
MonitoringReport
conflicting_label
compared_label
max_consecutive_conflicting_labels
conflicting_label_threshold
max_consecutive_conflicting_labels_threshold
DatabaseConnector
Label
<<use>>
LabelStorer
store_label ()
LabelManager
count_labels ()
store_label ()
MonitoringReportGenerator
generate_report ()
1
MonitoringSystemController
run ()
<<use>>
MonitoringSystemConfiguration
monitoring_window_lenght

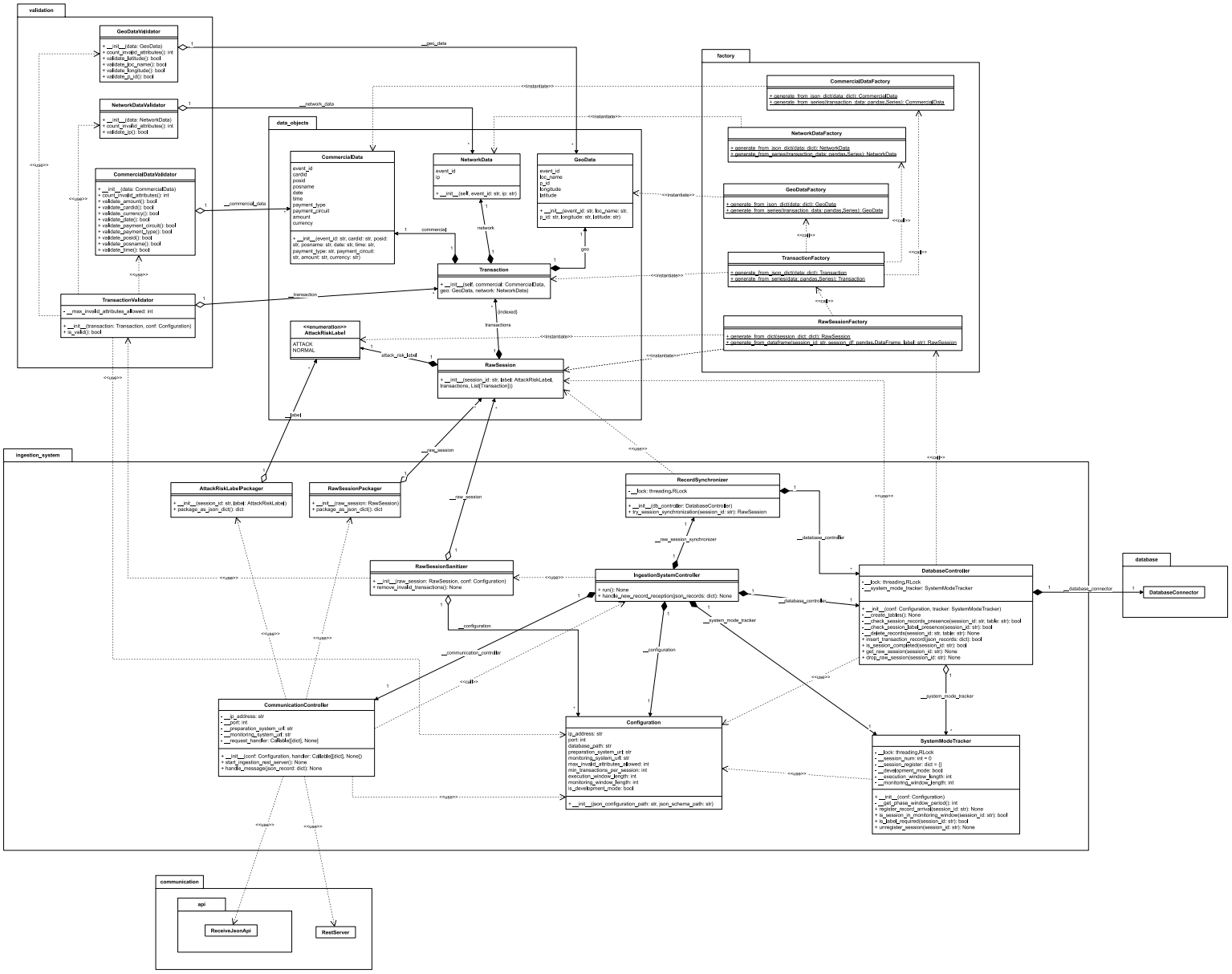
Workflow Design

Common packages (communication and database)

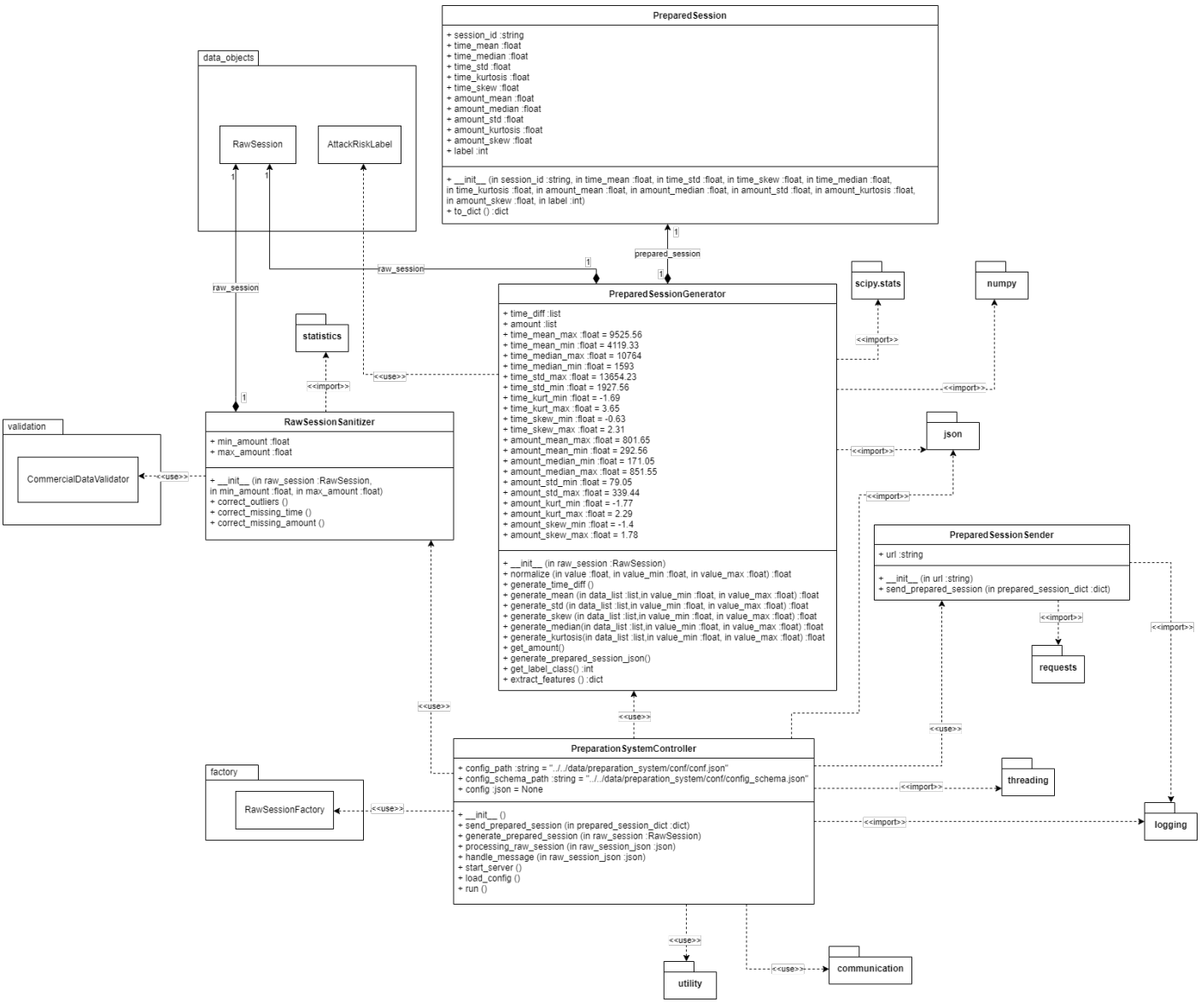




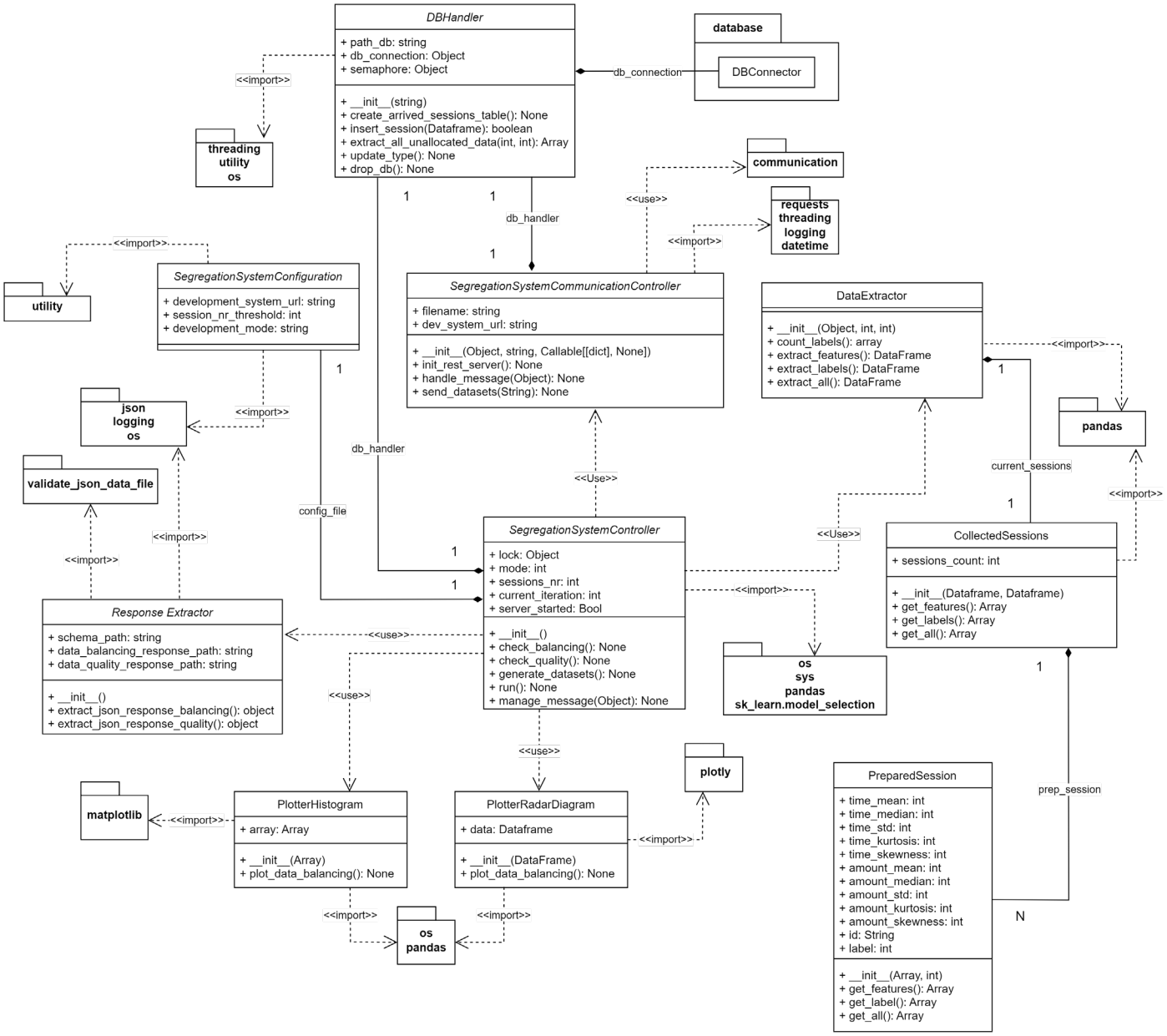
Ingestion System



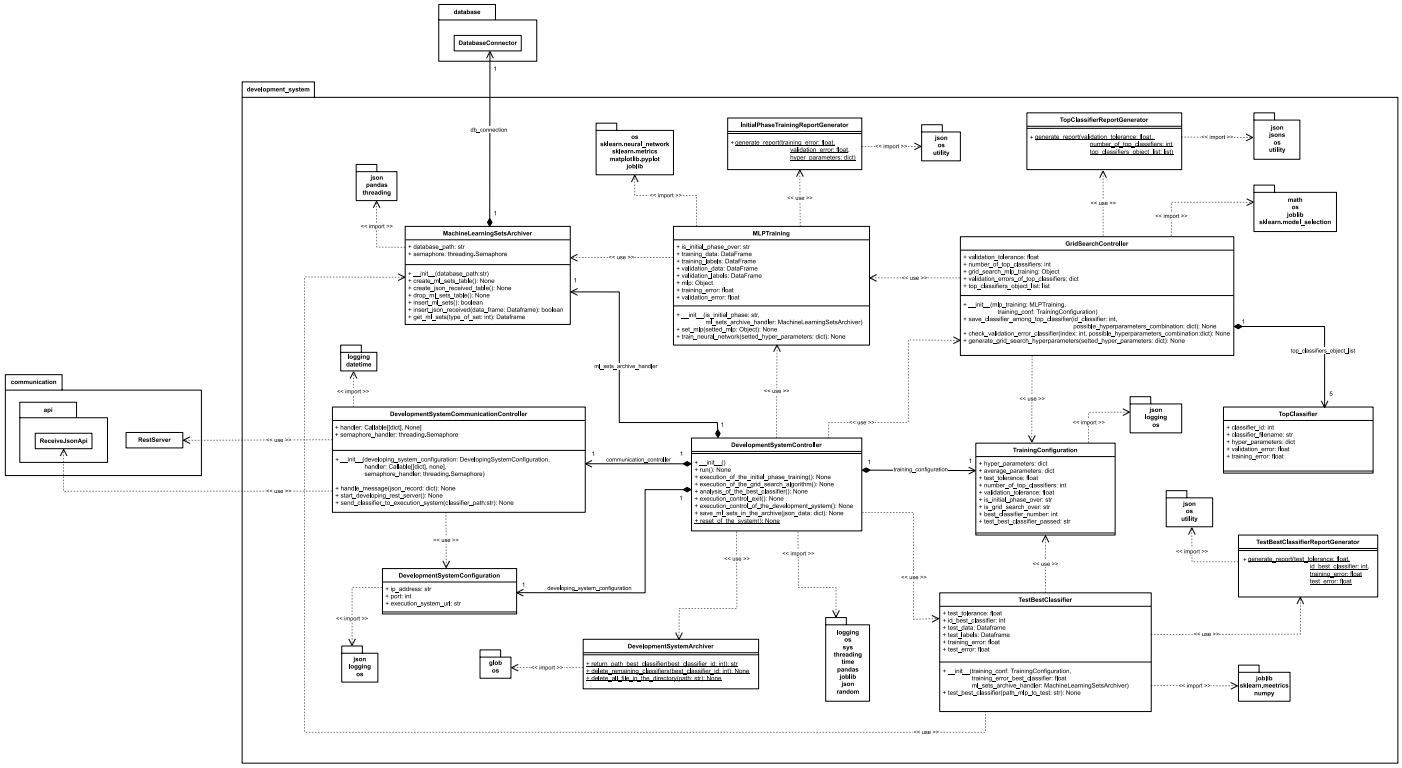
Preparation System



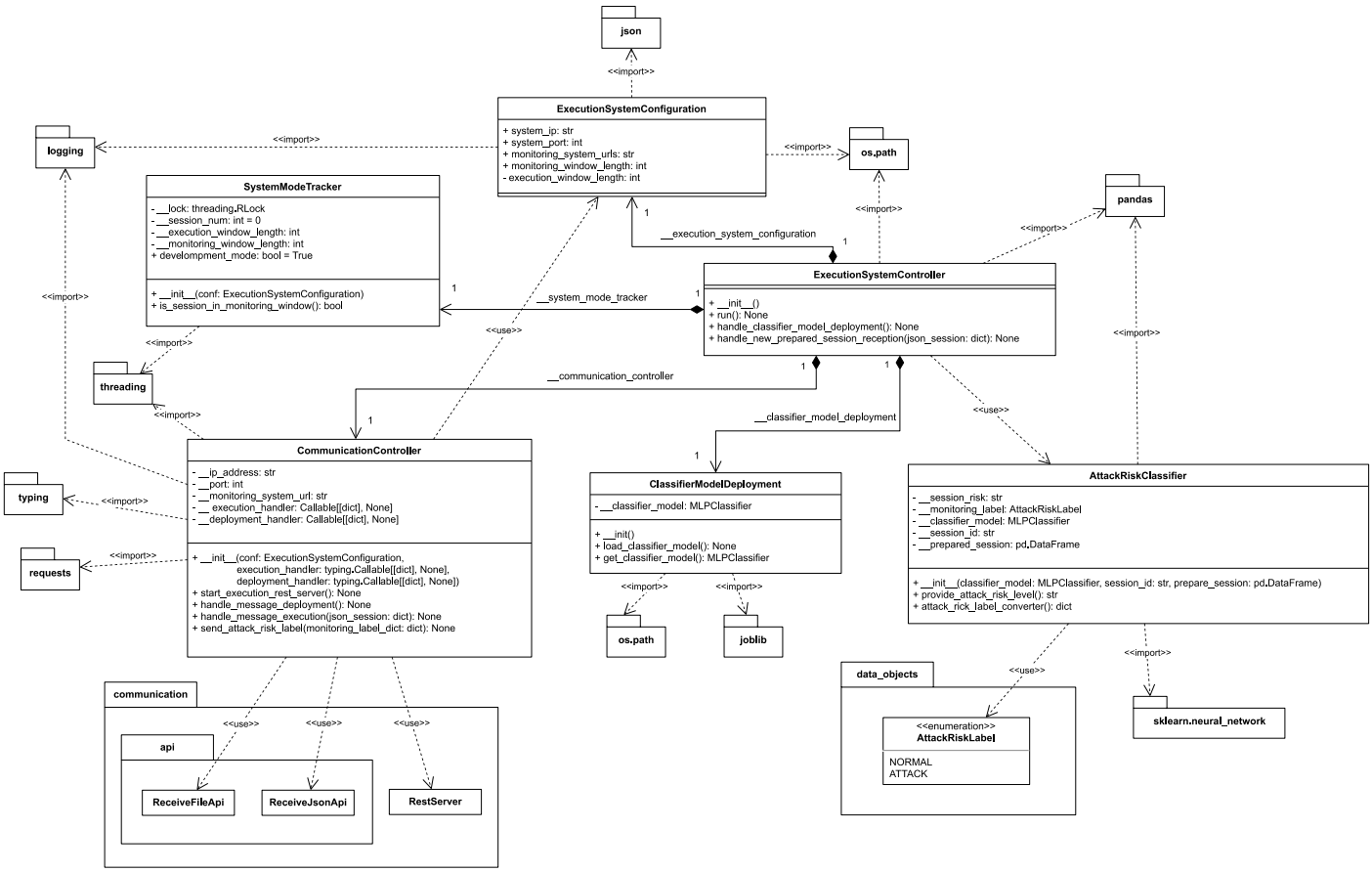
Segregation System



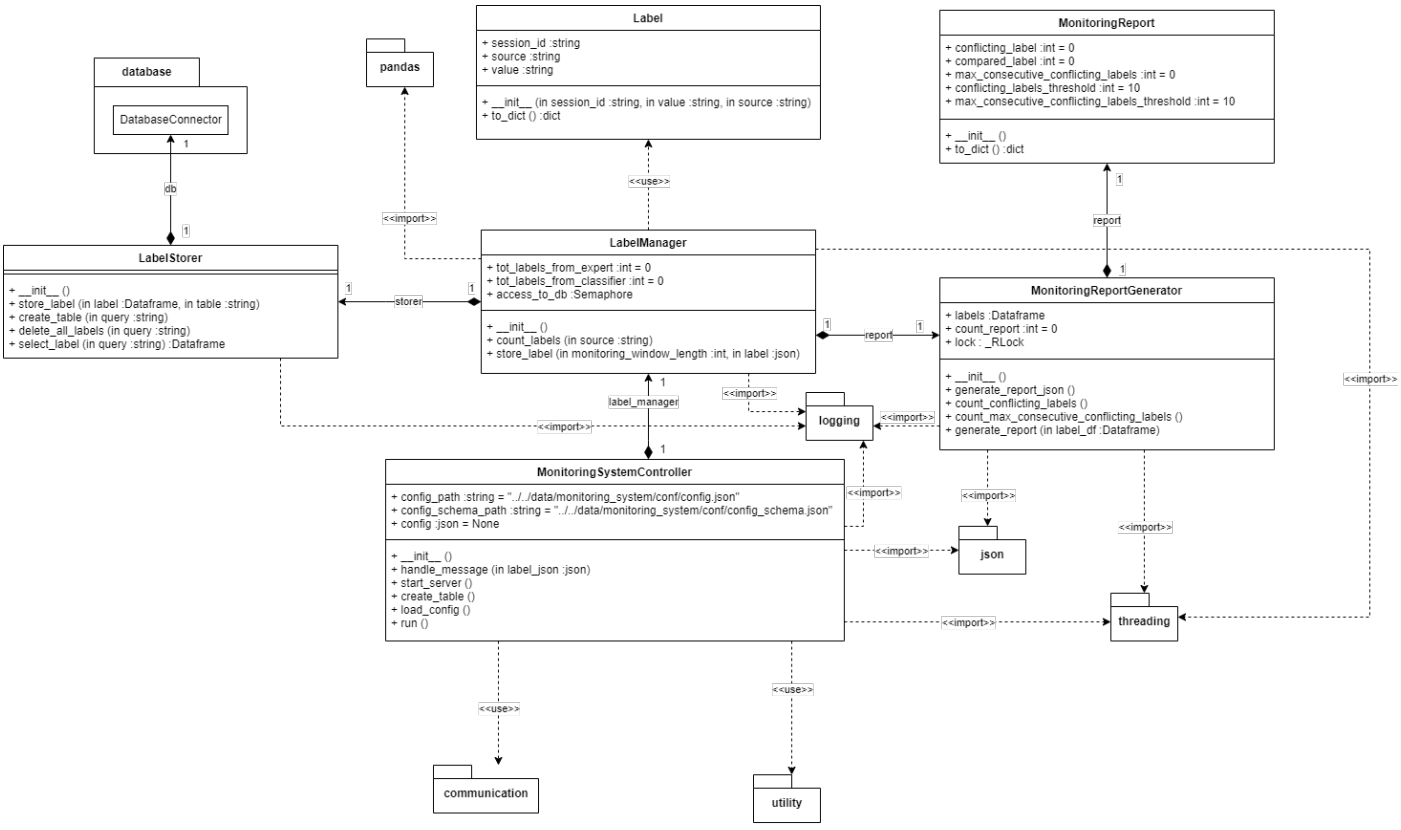
Development System



Execution System

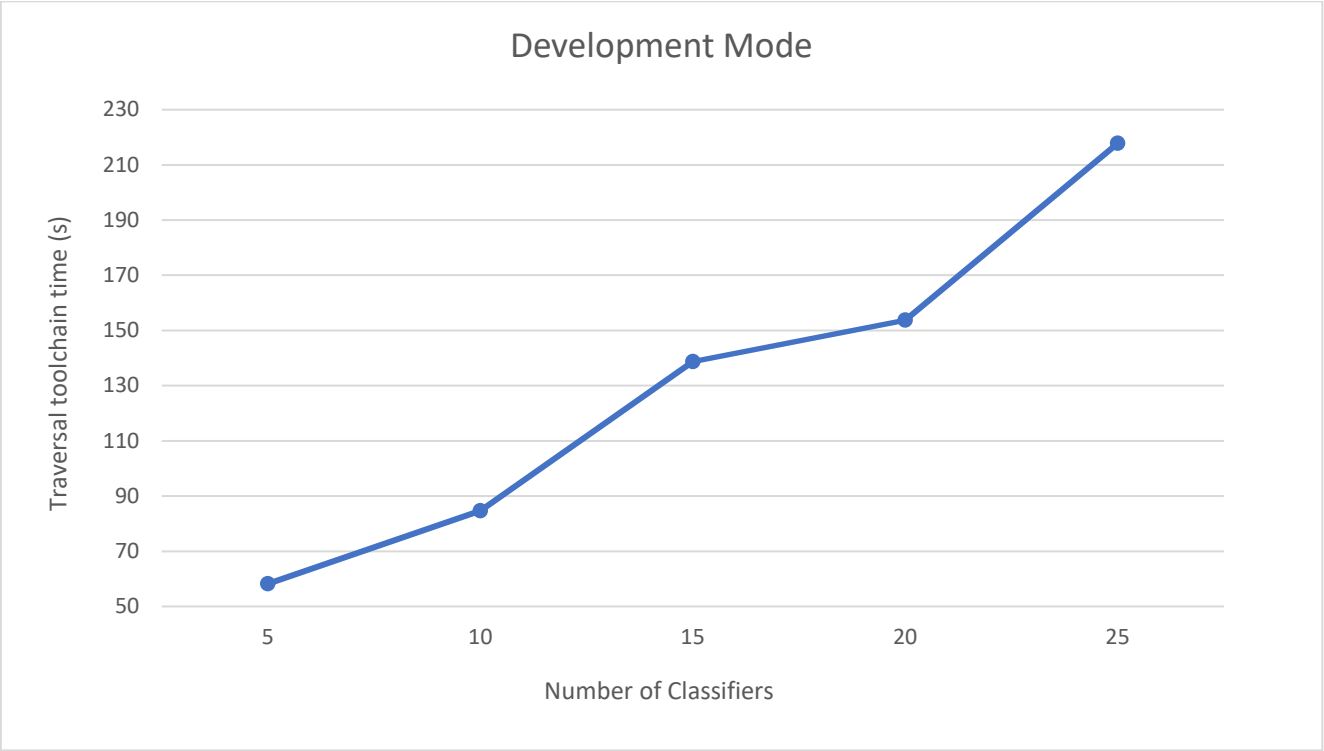


Monitoring System



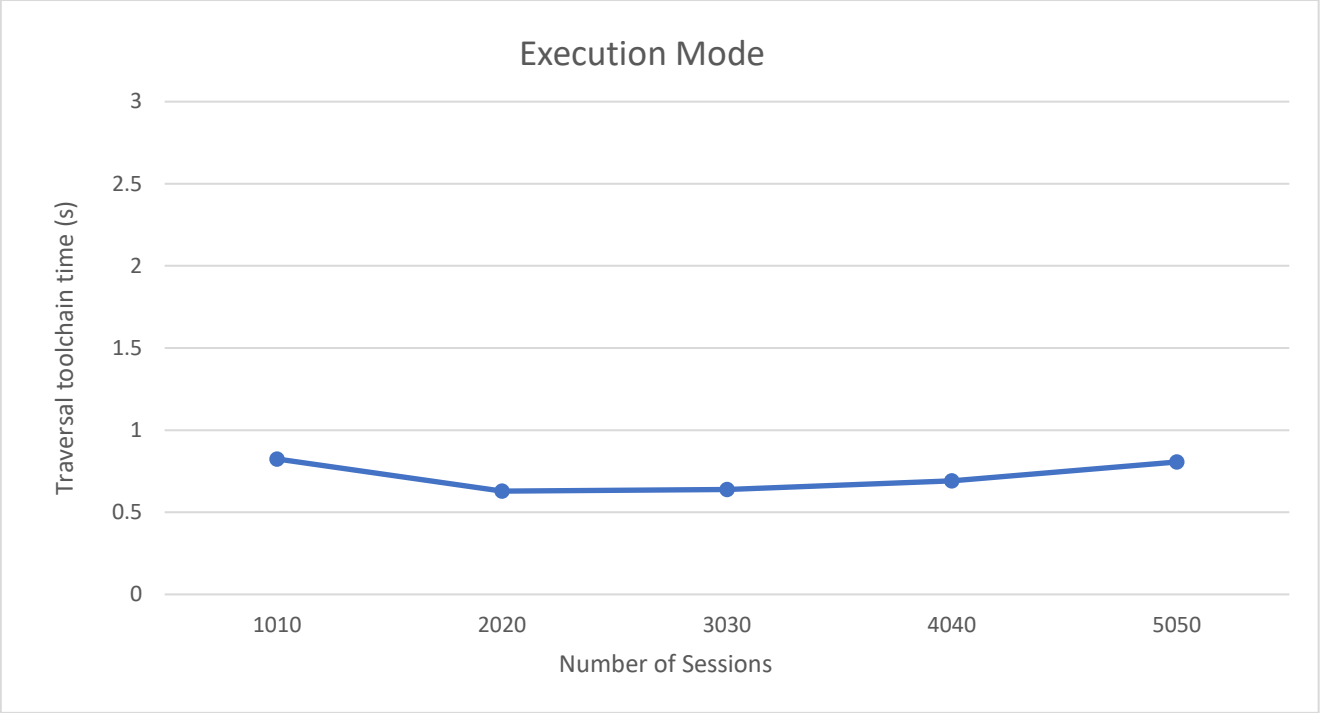
Testing

The two metrics analyzed during the testing phase are the non-responsiveness and the non-elasticity. The two metrics were examined in both modalities (development and execution). The development mode only considers the first part of the toolchain until the trained model is created, while the execution mode assumes the presence of an already trained model and then simulates the normal pipeline execution flow.



For the testing phase in development mode, a variable number of classifiers to be trained was considered.

In this case, a linear trend can be observed. The bottleneck system of the toolchain are the development system, since it trains and deploys the classifiers sequentially, whereas the other systems handles the input data in a parallel fashion.



For the testing phase in execution mode, a variable number of sessions to be classified was considered.

In this case, a constant trend can be observed. This is due to the fact that the execution system is multi-threaded, so it classifies the sessions in parallel.