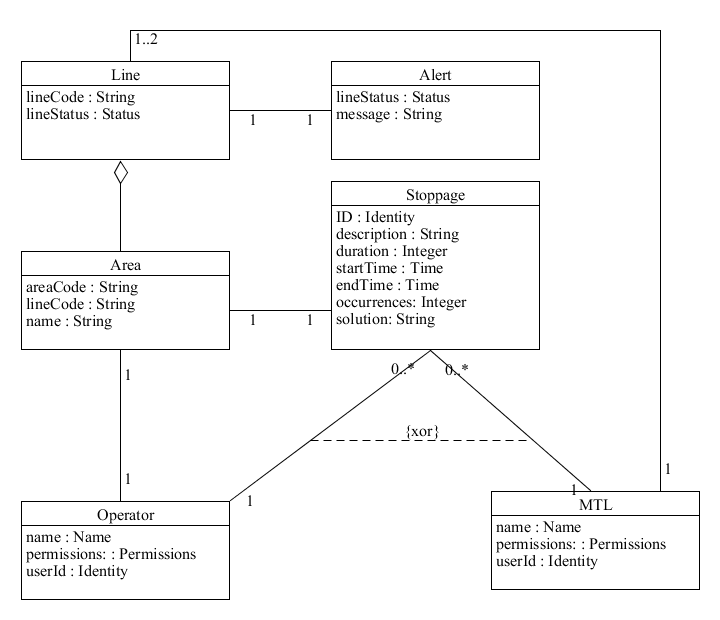
**Appendix 9: Analysis model**

To make transition between analysis and design it is necessary to add details to the conceptual model. It is done by splitting the original model into several class models supporting each of the use cases produced at the earlier stage. It helps to identify inconsistencies in conceptual model and to specify constraints on classes, attributes and associations.



**Figure 1** Class model supporting UC1 and UC5.

Invariants for classes and association loops:

**context** Alert **inv:**

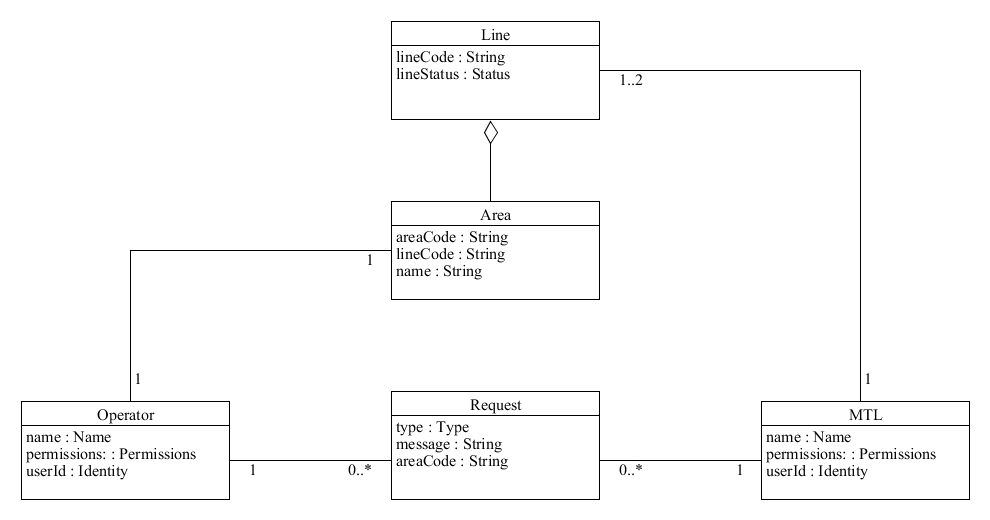
self.lineStatus = self.line.lineStatus

**context** Stoppage **inv:**

self.startTime < self.endTime

**context** Stoppage **inv:**

self.operator.area -> **includes**(self.area) or self.mtl.area -> **includes**(self.area)



**Figure 2** Class model supporting UC2.

Producing the model above uncovered a missing attribute for the *Request* class. The new attribute *areaCode* should guarantee independency of the class. Moreover, it is now worth to consider the suitability of this class for supporting other use cases. In particular, UC8 and UC9 will not be supported appropriately. At the later stage it might be necessary to rethink these two use cases, especially the need for making any requests when adding or removing users.

Invariants for classes and association loops:

**context** Operator **inv:**

self.line.lineCode = self.line.lineCode.mtl

**context** Operator **inv:**

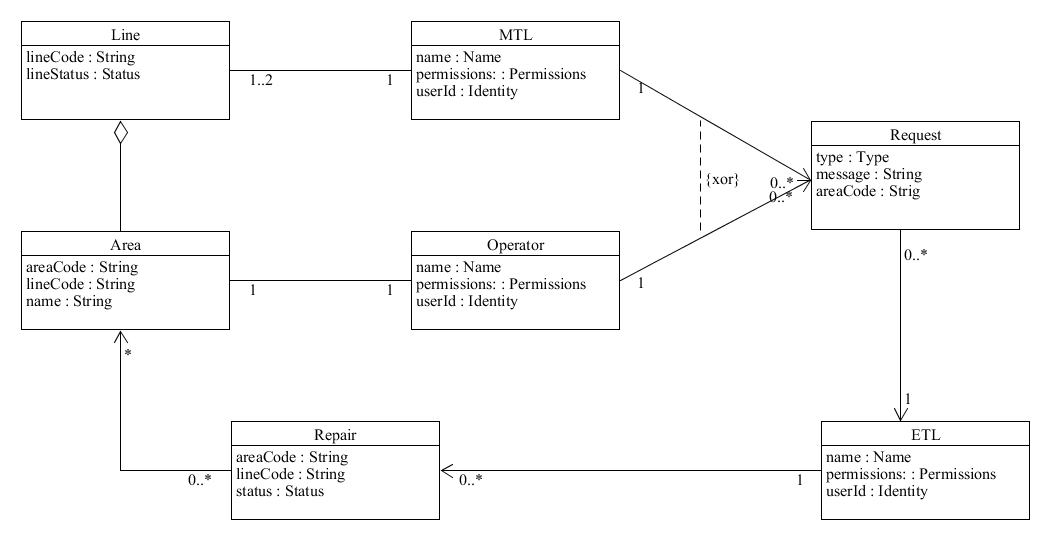
self.request = self.request.mtl

**context** Operator **inv:**

self.request -> **includes**(self.area)

**context** MTL **inv:**

self.request -> **includes**(self.line.area.operator **and** self.request.operator)



**Figure 3** Class model supporting UC3.

The class model above introduces navigation which allows tracking of the steps within the considered use case.

Invariants for classes and association loops:

**context** ETL **inv:**

self.request = self.request.operator **or** self.request = self.request.mtl

**context** ETL **inv:**

self.repair -> **includes**(self.repair.area)

**context** Operator **inv:**

self.request -> **includes**(self.area)

(new association between *MTL* and *Area*)

**context** MTL **inv:**

self.request -> **includes**(self.area)

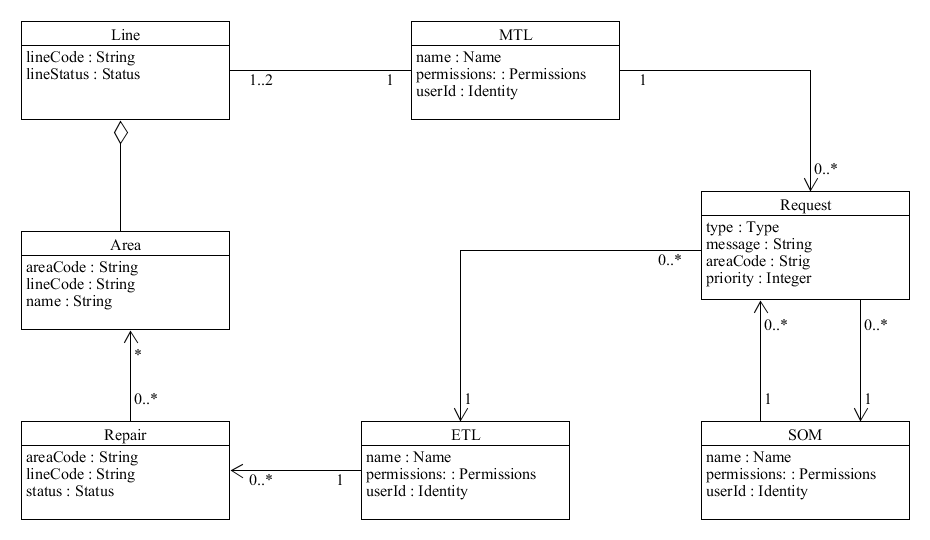
**context** Repair **inv:**

self.notEmpty **implies** self.etl.request.notEmpty

(new association between *Request* and *Area*)

**context** Request **inv:**

self.area = self.mtl.area **or** self.area = self.operator.area



**Figure 4** Class model supporting UC4.

The class diagram for UC4 required further changes in the *Request* class. The new attribute *priority* has been introduced due to the requirement for distinct urgency of the request depending on the user’s role. The MTL can request repair but if there are no available engineers it might be postponed by the ETL. The manager will assess the importance of continuing production within the line that broke down and might request an immediate action.

Invariants for classes and association loops:

**context** Request **inv:**

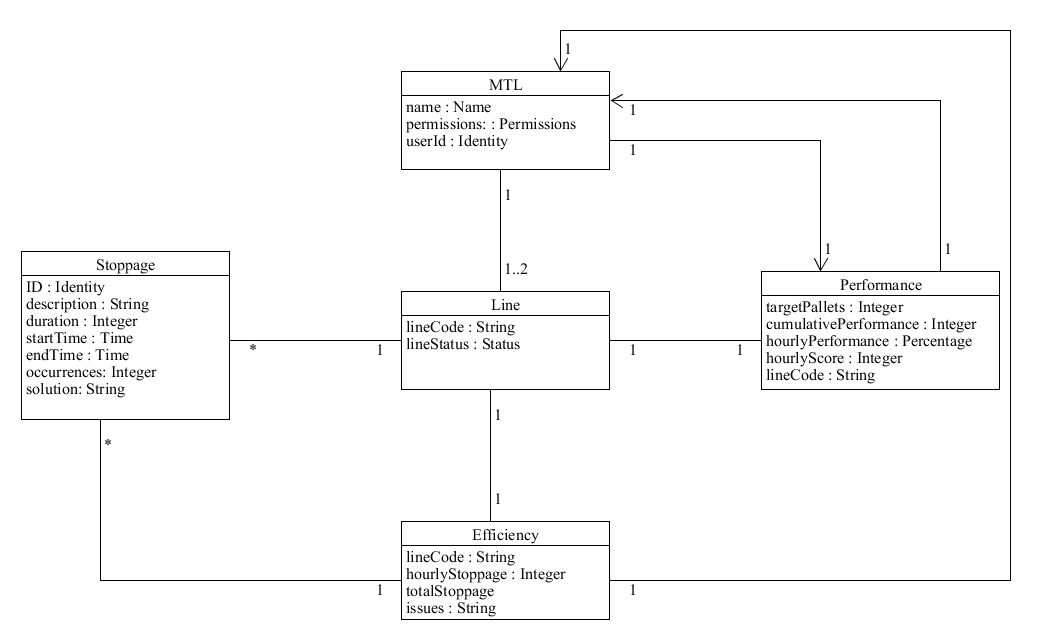
self.mtl = self.som

**context** Request **inv:**

self.area = self.mtl.area

**context** Request **inv:**

self.som = self.etl



**Figure 5** Class model supporting UC6.

Producing the diagram above identified the new associations. The performance and efficiency data is for a particular line therefore the both have to be associated with the *Line* class. Also, *Line* needs to be linked to *Stoppage* because there might be many stoppages within the given line and *Efficiency* and *Performance* are for the whole *Line* not like in the original conceptual model for the *Area*. The link between the *MTL* and *Performance* must be bidirectional because the *MTL* provides but also reviews performance data.

Invariants for classes and association loops:

**context** Line **inv:**

self.efficiency -> **includesAll**(self.stoppage.duration)

**context** Performance **inv:**

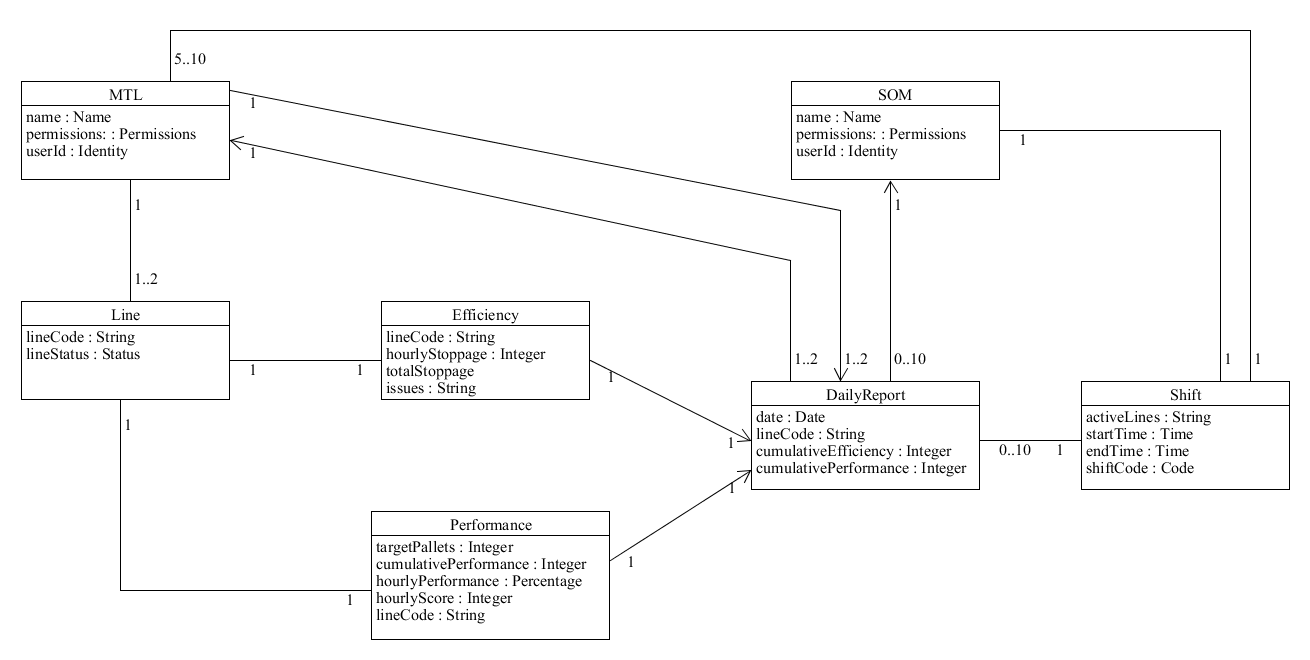
self.cumulativePerformance >= self.hourlyPerformance

**context** Efficiency **inv:**

self.issues.notEmpty **implies** self.stoppage.duration > 0

**context** Efficiency **inv:**

self.totalStoppage >= self.hourlyStoppage



**Figure 6** Class model supporting UC7.

The class model above revealed that the use case requires some changes. It is the *MTL’s* responsibility to amend details for *DailyReport* if no amendments are made the report is generated automatically. *SOM* will only need to view or print the report which is covered in the separate use case (UC10).

Invariants for classes and association loops:

**context** DailyReport **inv:**

self.cumulativeEfficiency = self.efficiency.totalStoppage

**context** DailyReport **inv:**

self.cumulativePerformance = self.performance.cumulativePerformance

(new attribute *date* for the class *Shift*)

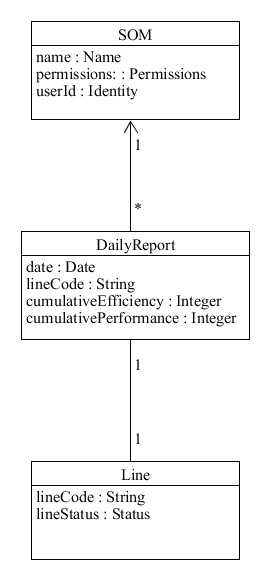
**context** DailyReport **inv:**

self.date = self.shift.date

**context** Shift **inv:**

self.som.dailyReport -> includesAll(self.dailyReport)

Due to the changes made in the *Request* class use cases 8 and 9 were discarded. *MTL’s* responsibilities do not include contacting system’s administrator and it is very unlikely that such request will be necessary. The new users are added when new person is employed. Therefore, addition or removal of the users will be outside the scope of the domain.



**Figure 7** Class model supporting UC10.

To support UC10 the new association between Line and DailyReport has been added. It represents the fact that for any one line there might be one daily report for a given date. Also, the SOM can review multiple daily reports hence the multiplicity has changed.

Invariants for classes and association loops:

**context** DailyReport **inv:**

self.lineCode = self.line.lineCode

For the reason that the analysis model shows the summary of changes made above it shall be enough listing the changes made rather than redrawing the whole diagram which will be cluttered version of detailed class models above.

Summary of changes:

* added new attribute *areaCode* to the *Request* class
* added association between *MTL* and *Area* (*MTL* can make requests for the *Area*)
* added association between *Request* and *Area* (*Request* can be made for support within the *Area*)
* added attribute *priority* to the *Request* class
* added association between *Performance* and *Line*
* added association between *Efficiency* and *Line*
* added association between *Stoppage* and *Line*
* removed association between *Stoppage* and *Area*
* added attribute *date* to the *Shift* class
* added association between *Line* and *DailyReport*
* modified multiplicities (as in the models above)

Changes in use cases

* removed UC8 and UC9
* modified UC7 (see below)

**Table 7** Textual representation of the (updated) *create daily report* use case.

|  |  |
| --- | --- |
| **Identifier and name** | UC7 *create daily report* |
| **Initiator** | *team leader* |
| **Goal** | The daily report is produced and can be printed out if required. |
| **Precondition** | The report can be produced 15 minutes before the end of the shift. The production line has been operating during the shift. |
| **Postcondition** | The system will summarise daily performance and efficiency. The report containing performance data will be produced. All the details will be saved for future reference. It will be possible to print out the report. |
| **Assumptions** | The expected initiator is team leader using web browser to perform the use case. There is a 15 minutes grace period to make amendments otherwise the system will produce the report automatically at the end of the shift. |
| **Main success scenario**  1 Team leader reviews the summary of daily performance and efficiency.  2 Team leader confirms that performance and efficiency data is correct.  3 Using the system team leader commit the report.  4 The system saves daily summary.  5 Team leader prints out the report.  **Extensions**  1.a.1 *Shift has ended*. The use case terminates, and the daily report is produced automatically by the system with the details provided during the shift.  2.a.1 *The performance data requires modification*. The team leader amends incorrect details. | |

**System operations**

The *Shift* class will be set as a system class. All operations represented by the use cases are performed on the *Shift*. The Shift is associated directly or indirectly with all other classes.

Use case 1

**contex** Shift::areaActive(lineCode : String, lineStatus : String) : Boolean

**context** Shift::activateDowntime(stoppage : Stoppage, operator : Operator, area : Area, line : Line, alert : Alert): String

**pre:**

- - no alert is linked to line

**post:**

- - if areaActive(lineCode : String, lineStatus : String) in line returned true then

- - a new Stoppage object, stoppage, will have been created (object creation)

- - operator will be linked to stoppage

- - area will be linked to stoppage

- - a new Alert object, alert, will have been created (object creation)

- - alert will be linked to line

- - a string indicating stoppage will have been returned

- - otherwise a string indicating failure will have been returned

Use case 2

**context** Shift::contactTeamLeader(request : Request, operator : Operator, line : Line, area : Area): Boolean

**pre:**

- - areaActive(lineCode : String, lineStatus : String) in line returned false

**post:**

- - a new Request object, request, will have been created (object creation)

- - request will be linked to operator

- - request will be linked to MTL

- - request will be linked to area

- - request will be linked to line

- - true will be returned

- - otherwise, false will be returned

**context** Shift::informTeamLeader(request : Request, stoppage : Stoppage, line : Line, area : Area): Boolean

**pre:**

- - stoppage duration is more or equal to 15 minutes

**post:**

- - a new Request object, request, will have been created (object creation)

- - request will be linked to MTL

- - request will be linked to area

- - request will be linked to line

- - true will be returned

- - otherwise, false will be returned

Use case 3

**context** Shift::makeRequest(type : String, message : String, areaCode : String) : String

**post:**

- - type String will have been returned

**context** Shift::requestRepair(line : Line, area : Area, operator : Operator, etl : ETL, repair : Repair, request : Request) : Boolean

**pre:**

- - areaActive(lineCode : String, lineStatus : String) in line returned false

- - makeRequest(type : String, message : String, areaCode : String) returned Repair

**post:**

- - a new Request object, request, will have been created (object creation)

- - request will be linked to operator

- - request will be linked to ETL

- - a new Repair object, repair, will have been created (object creation)

- - area will be linked to repair

- - true will be returned

- - otherwise, false will be returned

Use case 4

**context** Shift::contactManager(stoppage : Stoppage, request : Request, mtl : MTL, som : SOM) : Boolean

**pre:**

- - stoppage duration is more or equal to 30 minutes

**Post:**

- - a new Request object, request, will have been created (object creation)

- - request will be linked to mtl

- - request will be linked to som

- - true will be returned

- - otherwise, false will be returned

Use case 5

**context** Shift::deactivateDowntime(stoppage : Stoppage, operator : Operator, line : Line, alert : Alert, time : Time) : Boolean

**pre:**

- - areaActive(lineCode : String, lineStatus : String) in line returned false

**post:**

- - stoppage endTime will have been set to time

- - stoppage duration will have been set to the difference between startTime and endTime

- - stoppage occurrences will have been updated

- - stoppage solution will have been updated

- - alert will be destroyed (object deletion)

- - areaActive(lineCode : String, lineStatus : String) in line will have returned true

- - true will be returned

- - otherwise, false will be returned

Use case 6

**context** Shift::activatePerformance(mtl : MTL, target : Integer, lineID : String, line : Line) : Boolean

**pre:**

- - mtl is linked to Line with lineCode eqal to lineID

**post:**

- - a new Performance object, performance, will have been created (object creation)

- - performance targetPallets will be set to target

- - performance hourlyScore will be set to 0

- - performance hourlyPerformance will be empty

- - performance cumulativePerformance will be empty

- - performance will be linked to line

- - mtl will be linked to performance

- - true will be returned

- - otherwise, false will be returned

**context** Shift::activateEfficiency(mtl : MTL, lineID : String, line : Line) : Boolean

**pre:**

- - mtl is linked to Line with lineCode eqal to lineID

**post:**

- - a new Efficiency object, efficiency, will have been created (object creation)

- - efficiency hourlyStoppage will be set to 0

- - efficiency totalStoppage will be set to 0

- - efficiency issues will be empty

- - efficiency will be linked to line

- - mtl will be linked to efficiency

- - true will be returned

- - otherwise, false will be returned

**context** Shift::updatePerformance(performance : Performance, mtl : MTL, score : Integer) : Boolean

**pre:**

- - performance must exist

- - performance is linked to mtl

**post:**

- - performance hourlyScore will be set to score

- - performance hourlyPerformance will be changed

- - performance cumulativePerformance will be changed

- - true will be returned

- - otherwise, false will be returned

**context** Shift::updateEfficiency(efficiency : Efficiency, stoppages : Collection, lineCode : String, time : Time) : Boolean

**pre:**

- - efficiency must exist

- - line code for stoppages equals lineCode

**post:**

- - efficiency issues will be set to three strings containing description of three stoppages with the highest duration

- - efficiency hourlyStoppage will be set to total duration of stoppages within 60 minutes prior to time

- - efficiency totalStoppage will be updated adding efficiency hourlyStoppage

- - true will be returned

- - otherwise, false will be returned

**context** Shift::checkPerformance(mtl : MTL, performance : Performance, line : String, efficiency : Efficiency) : Collection

**pre:**

- - mtl is linked to performance

- - mtl is linked to efficiency

- - performance lineCode equals line

- - efficiency lineCode equals line

**post:**

- - performance hourlyPerformance will be added to Collection

- - performance cumulativePerformance will be added to Collection

- - efficiency hourlyStoppage will be added to Collection

- - efficiency totalStoppage will be added to Collection

- - Collection will be returned

Use case 7

**context** Shift::makeDailyReport(mtl : MTL, performance : Performance, line : Line, efficiency : Efficiency, lineID : String, date : Date) : Collection

**pre:**

- - mtl is linked to performance

- - mtl is linked to efficiency

- - performance is linked to Line

- - efficiency is linked to Line

- - date equals Shift date

**post:**

- - a new DailyReport object, report, will have been created

- - report will be linked to performance

- - report cumulativePerformance will be set to performance cumulativePerformance

- - report cumulativePerformance will be added to Collection

- - report will be linked to efficiency

- - report cumulativeEfficiency will be set to efficiency totalStoppage

- - report cumulativeEfficiency will be added to Collection

- - report lineCode will be equal to lineID

- - Collection will be returned

Use case 10

**context** Shift::isReport(aDate : Date, lineID : String) : Boolean

**pre:**

- - DailyReport object with date equal to aDate exist

- - DailyReport object’s lineCode equals lineID

**context** Shift::reviewPerformance(som : SOM, lineID : String, aDate : Date, report : DailyReport) : Object

**pre:**

- - isReport(aDate : Date, lineID : String) must have returned true

**post:**

- - som will be linked to report

- - report will be linked to line

- - DailyReport object will be returned

Further analysis uncovered several inconsistencies within the use cases, classes and associations. The majority of them were resolved dynamically while specifying system operations. More changes might be required at the design stage. The most problematic issues are related to the class models. Especially, Stoppage class does not work very well while dealing with the past and current stoppages. Therefore, it might be necessary to introduce new classes which would better represent the system’s behaviour. Stoppage could become a parent class to two child classes or could be an abstract class. Some of the use cases were not specific enough for convenient translation into system operations. However, for the purpose of prototyping the models created so far should be sufficient.